TECHNISCHE UNIVERSITÄT
CHEMNITZ

Developing a Car2X Communication Application using a Queriable Wireless Sensor Network

Master Thesis

For the fulfillment of the Academic Degree
M.Sc. in Automotive Software Engineering

Dept. of Computer Science
Chair of Computer Engineering

Submitted by: Srinivasu Jitta
Student ID: 357928
Submission date: 07.02.2018

Supervisors: Prof. Dr. Wolfram Hardt
Dipl.-Inf. René Bergelt

Abstract

The development of wireless sensor networks has reached a point where each individual node of a network may store and deliver a massive amount of (sensor-based) information at once or over time. In the future, massively connected, highly dynamic wireless sensor networks such as vehicle-2-vehicle communication scenarios may hold an even greater information potential. This is mostly due to the increase in node complexity. Consequently, data volumes will become a problem for traditional data aggregation strategies traffic-wise as well as with regard to energy efficiency.

Therefore, in this thesis, proposed a database aggregation strategy can be used to minimize most of these problems of big data in embedded and wireless sensor networks which enables the efficient use of energy and the handling of large data volumes.

Moreover, evaluated latency and traffic volume in the network based on experiments using sensor platforms.

Keywords: wireless sensor networks, query languages, data streaming, IoT, car2x

Acknowledgement

This report is written during Master thesis at the Faculty of Computer Science, Department of Computer Engineering at Technische Universität Chemnitz.

I would like to thank Prof. Dr. Wolfram Hardt for giving me the opportunity to work in his department on challenging research project with topic "Developing a Car2X Communication Application using a Queriable Wireless Sensor Network", which allow me to pursue different ideas, eventually leading to this thesis report. It was great opportunity to work in Car2X technology and PLANetary query system. I would like to express my gratitude to several individuals who supported in completion of thesis project, Dr. Ariane Heller, she always helps in solving organizational issues raised at each milestone.

Finally, I would like to thank my supervisor Dipl.-Inf. René Bergelt, for providing valuable review and feedback comments, the time he spent for discussing problems and new idea for achieving goals of the project.

Last but not the least, I would like to express my heart-felt gratitude to my family members and my friends for their support and encouragement.

Contents

List of Figures

List of Tables

List of Abbreviations

WSN Wireless Sensor Networks

IoT Internet of Things

SQL Structured Query Language

DARPA United States Defense Advanced Research Projects Agency

DSNs Distributed Sensor Networks

QoS Quality of service

SON Self-Organizing Networks

CDS Connected Dominant Set

OS Operating System

SNQP Sensor Network Query Processors

VANET Vehicular Ad-hoc NETwork

ITS Intelligent Traffic Systems)

V2I Vehicle to Infrastructure

V2V Vehicle to Vehicle

MAC Medium Access Control

ADAS Advanced Driver Assistance Services

WAVE Wireless Ability in Vehicular Environments

ACC Adaptive Cruise Control

BLE Bluetooth Low Energy

WPAN Wireless Personal Area Network

FFD Full Function Devices

RFD Reduced Function Device

AP Access Points

STA Stations

P2P Peer-to-Peer

WFA Wi-Fi Alliance

GO Group Owner

NAT Network Address Translation

WPS Wi-Fi Protected Setup

DHCP Dynamic Host Configuration Protocol

PBC Push Button Control

VSN Vehicular Sensor Network

WAVE Wireless Access in the Vehicular Environment

CSMA/CA Carrier Sense Multiple Access/Collision Avoidance

RSU Road Side Unit

OBU On-Board Unit

AODV Ad-Hoc On-Demand Distance Vector

DSR Dynamic Source Routing

ABR Associativity Based Routing

TORA Temporally Ordered Routing Algorithm

DSDV Destination-Sequenced Distance Vector

ZRP Zone Routing Protocol

RREQ Route REQuest

RREP Route REPly

RERR Route ERRor

1 Introduction

1.1 Background

Highly sophisticated technologies have been included in modern vehicles to provide active safety. For instance, Driver assistance systems like Electronic Stability Control (ESC), Adaptive Cruise Control (ACC), Active braking, Pedestrian Recognition or Night Vision are used to improve future road safety. All these systems actions depend on information received from their local sensors, which are Sonar, Camera, Radar or Lidar [1]. To detect nearby cars, Park Assistant System uses ultrasonic sensors, where measurement range is about 4 m. The camera-based object detection algorithms are used automotive cameras to detect cars or pedestrians up to 80 m distance [1]. The transmission range of Radar or Lidar is up to 200 m, which allows active braking even at high speeds and the Lidar-based system is used to detect snowfall or heavy rain and gives important information to the Electronic Stability Control (ESC). Figure 1.1 shows Range comparison of cars sensors.

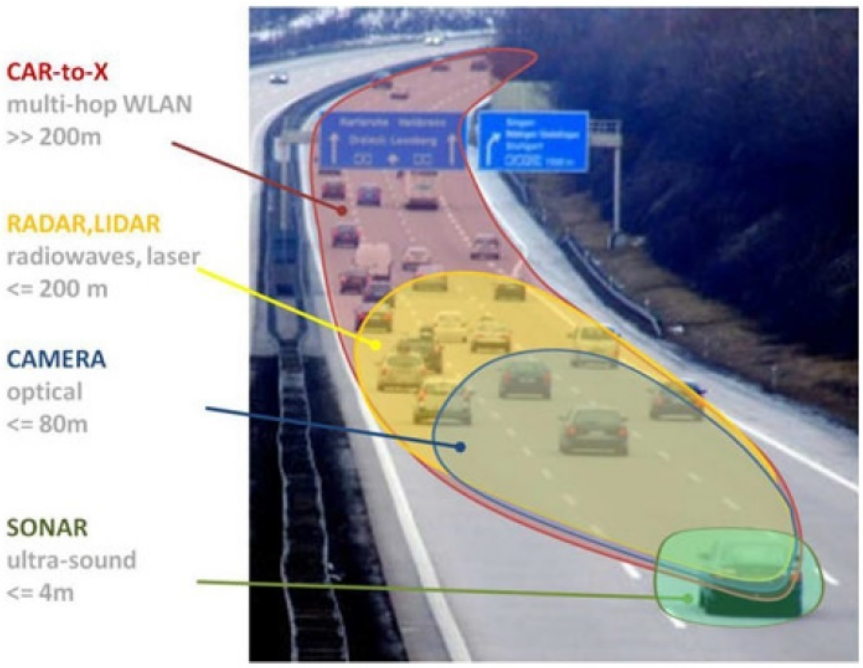


Figure 1.1: Vehicle sensor range comparison [1]

The Car2X communication technology offers new possibilities for improving traffic efficiency and active safety at a large scale [1]. The warnings of critical situations are not restricted to local detection and once a risky situation is detected and then forwarded via multiple hops using Car2X communication. Therefore, the approaching car drivers may respond in time and adjust their driving behavior [1].

Car2X communication is based on the IEEE 802.11p or WAVE (Wireless Ability in Vehicular Environments) standard, which enables time-critical safety applications at very low data transmission delay. The European Commission has started the allocation of the 30 MHz spectrum, from 5.875 to 5.905 GHz bandwidth for vehicular communication and according to the European profile of 802.11p, three channels with 10 MHz each, are assigned [25].

The number of sensors in the vehicle has increased drastically due to the various comfort and safety applications [26]. Because of the internal structure of the vehicle, the wired architecture is not flexible and scalable enough [27]. Hence, there is an increasing level of appeal to design a system in which wireless links had replaced by the wired connections in the sensor networks [26].

Presently, research focuses mainly on some big data projects deal with large amounts of data in distributed, embedded systems and wireless sensor networks. Consequently, different approaches are needed for different applications. Due to the increase in node complexity in vehicles, traditional aggregation strategies reach their limits with respect to data traffic and efficient use of energy [28]. With recent developments, such as Car-2-Car(C2C) or Car-2-Roadside communication [29], a large amount of sensor data is generated and needs to be collected together and evaluated [30]. Therefore, the importance of handling big data volumes in embedded, and WSNs in an efficient way is increasing dramatically.

1.2 Problem Statement

The database aggregation strategy should aim to handle large volumes of data concerning traffic-wise as well as energy efficiency.

1.3 Thesis Outline

The thesis outline provides a short introduction of the chapters included in the report.

Chapter Introduction provides the introduction of different sensors in vehicles. Presents a short description of Car2X and problems regarding WSNs in Car2X.

Chapter State of the Art introduces the fundamentals of Wireless Sensor Networks, Query processing in WSN and Car2X communication. This chapter contains a brief description of the WSN application, deployment process, topologies, query processing architecture, Car2X architecture and application domain.

Chapter Concept Design presents relevant work done by Wi-Fi Alliance, Bluetooth Special Interest Group, ZigBee within the field of wireless communication technologies. Introduces the fundamentals of Bluetooth, ZigBee module, IEEE 802.11 WLAN, Wi-Fi Direct, and functionalities of Wi-Fi Direct.

Introduces the PLANetary library, which was developed at the Professorship for Computer Engineering at Technische Universität Chemnitz.

Presents Car2X communication uses cases to implement and required sensors with their values.

Chapter System Architecture and Implementation This chapter presents a detailed description of the implementation and presents the tools and environment used.

Presents implementation process of wireless communication between Raspberry Pis using Wi-Fi Direct.

Analyzing PLANetary libraries and implementation of PLANetary node.

The implementation process of Car2X communication scenario using PLANetary.

Chapter Performance Evaluation presents an analysis of query latency for each node.

Chapter Conclusion summarizes the key elements from previous chapters and presents conclusions.

2 State of the Art

This state of the art chapter gives an Introduction to Wireless Sensor Networks (WSN), Query processing in WSN and Car2X communication. This research mainly focuses on the bridge between WSN and Car2X communication. Explaining different Query approaches, Query systems, and Car2X communication scenarios.

2.1 Introduction to Wireless Sensor Networks

The development of WSNs was inspired by military applications, especially surveillance in dangerous zones [31]. The United States Defense Advanced Research Projects Agency (DARPA) followed up the Distributed Sensor Networks (DSN) program for the US military in 1980, because of that started research on WSNs. Presently, WSNs have been one of the most useful and important technologies in the 21st century [2].

2.1.1 Wireless Sensor Networks

A Wireless Sensor Network (WSN) is a network formed by a large number of sensor nodes [32]. Each sensor node has sensors, which detects or gathers information from environmental and physical conditions such as light, temperature, sound, and pressure. Sensor nodes collect the data and share it with other sensor nodes [33].

WSNs are named as one of the best information gathering methods to build the information and communication system which improves the reliability and efficiency of infrastructure systems significantly [2]. Compared to the wired solution, easiest deployment and very flexibility of devices in WSNs [2].

Due to increase in the technological development of sensors, WSNs will become the key technology for IoT (Internet of Things). WSNs consist of the Sensor nodes/Actuator nodes, Gateway, and Client where the Gateway is to collect the data from sensor nodes then combines them. When the application needs the data, the gateway sends data to the client, which manages and configures WSNs. The structure of a WSN is shown in Figure 2.1

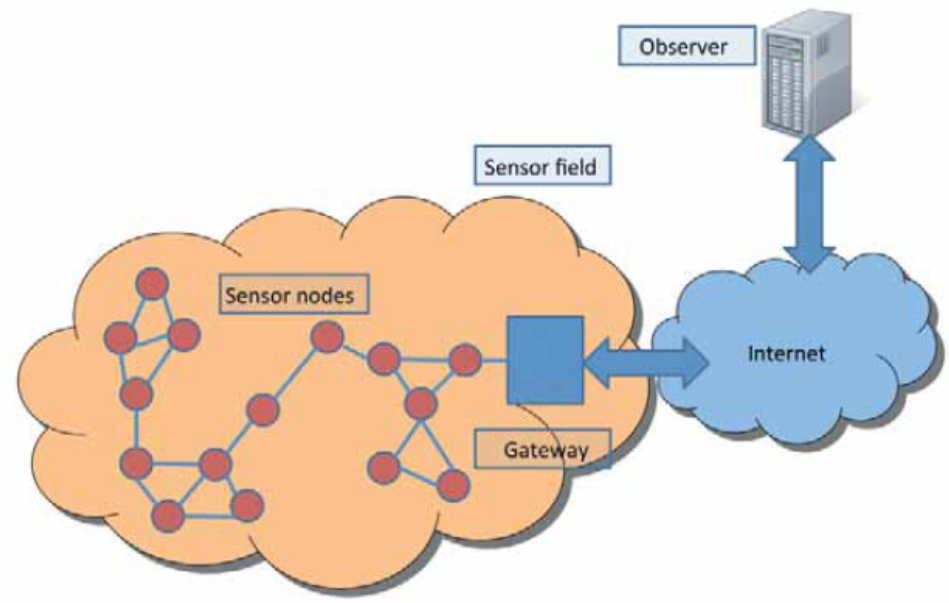


Figure 2.1: Wireless Sensor Networks [2]

2.1.2 WSN Applications and Examples

Recently, WSNs become popular tools [34] with the fast development of sensing and wireless communication technology, which drops the cost of WSNs rapidly and increases the capabilities. The applications are expanding from the military area to commercial and industrial fields.

Some of the applications are Industrial infrastructure, Automation, Military, Traffic, and Healthcare.

According to ON World, Figure 2.2 shows the revenue growth of WSNs in industrial infrastructure (From 2011–2017 in Million $) all over the world. Growth has been increasing every year.

WSN application examples

Some of the WSN application examples are disaster relief operations, where sensor nodes are dropped from an aircraft over a wildfire, and then each node measures temperature because of that it is easy to map the temperature and take up necessary relief activities [35]. In biodiversity mapping, sensor nodes are used to observe the life of animals. In intelligent buildings, energy wastage is reduced by proper ventilation, air conditioning, humidity control and measuring room temperature and air flow [36].

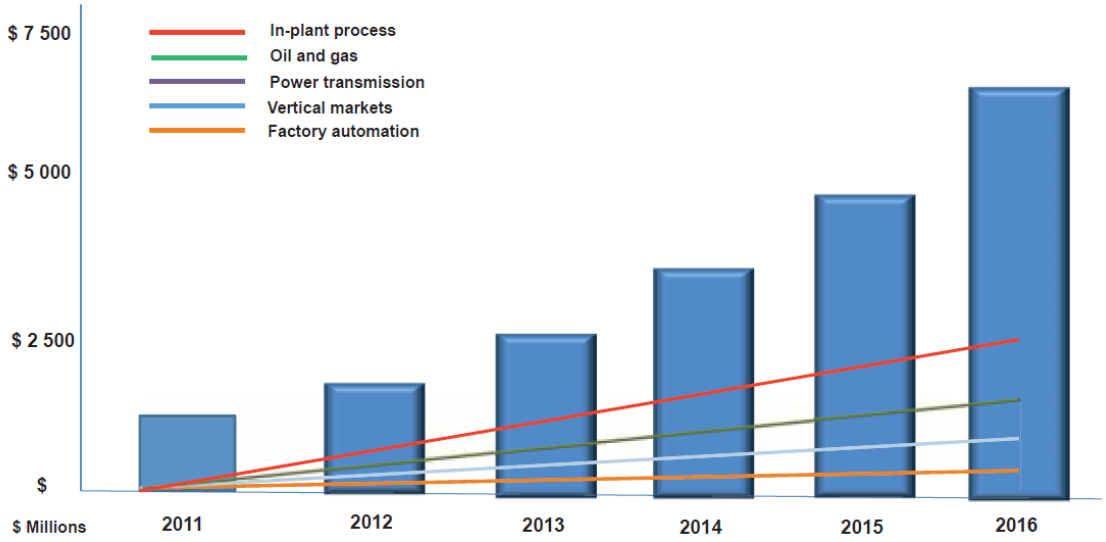


Figure 2.2: WSN revenue growth in industrial infrastructure [2]

Characteristic requirements for WSNs

The characteristic requirements for WSNs are explained below.

Quality of service: Traditional QoS methods cannot be applied, however, service of WSNs must provide right answers at the right time [36].

Fault tolerance: Must be robust, when the node failure occurs (sensors has low energy).

Lifetime: The network should have done their role until it runs out of energy, where the lifetime of individual nodes is not so important. However, care for all nodes should be equivalent.

Scalability: Support a large number of nodes [37].

Wide range of densities: Small number of nodes per unit sensor field, and depends on application.

Programmability: Sometimes to improve flexibility, re-programming of nodes in the sensor field might be needed.

Maintainability: WSN must adapt to changes, self-monitoring, adapt operation and must merge with the newly added nodes in the network.

2.1.3 WSN Deployment Process

WSN consists of a large number of sensor nodes [38].

The general deployment process of self-organizing networks is shown in Figure 2.3.

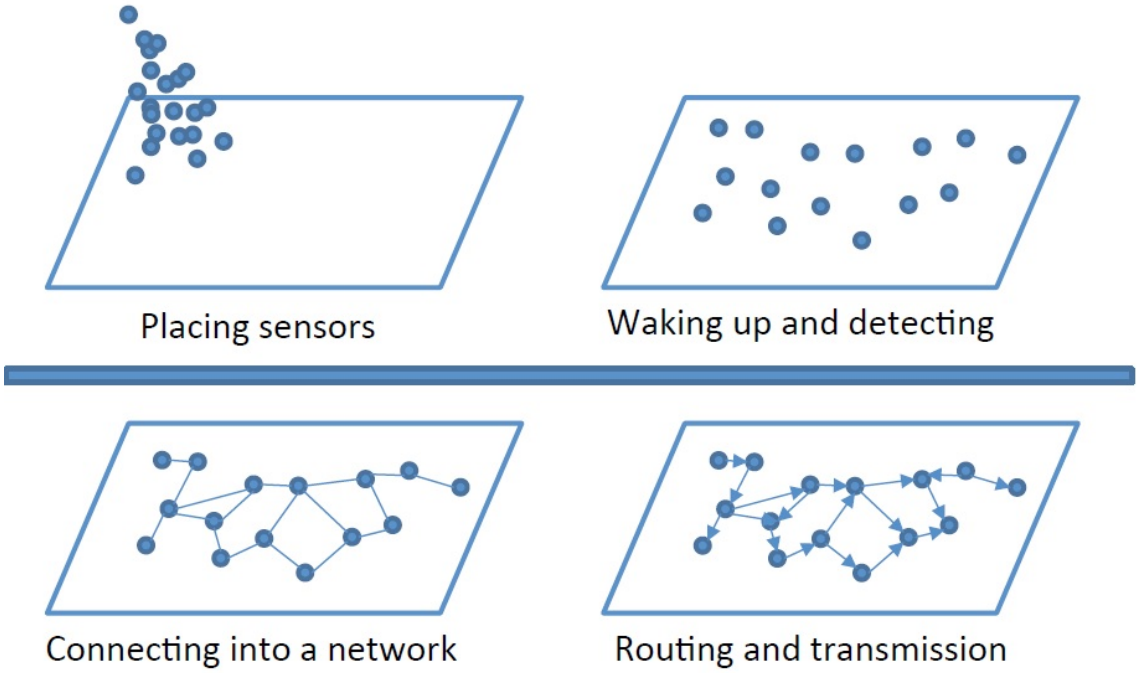


Figure 2.3: Deployment process of WSN [2]

Firstly, the sensor nodes send their status to the other nodes and receive status from other nodes to communicate with each other.

Secondly, according to a specified topology (linear, star, tree, and mesh), the sensor nodes are prearranged into a connected network.

Finally, to transmit the sensing data on the constructed network, suitable routes needed to be computed.

Batteries usually power the sensor network nodes, because of that the transmission distance of WSN nodes is short and depends on the actual environment. It can be up to 10 to 50 meters in the open outdoor environment. Due to the walls in indoor environments, transmission or signal strength will be weakened. Therefore, it will

be less in the indoor environment compared to outdoor.

To increase transmission distance or the coverage of a network, sensor networks use appropriate routing techniques depending on the topology. Hence, sensor network nodes act as both transmitter and receiver [2].

To transmit the data to the Gateway, the source node (first node in the sensor network) sends the data to a nearby node, and then it forwards the data to one of its neighboring nodes that are on the route towards the gateway. Until the data arrives at the gateway, repeats the forwarding.

2.1.4 Topologies

Figure 2.4 shows the topologies of WSNs, which are a star, tree, and chain.

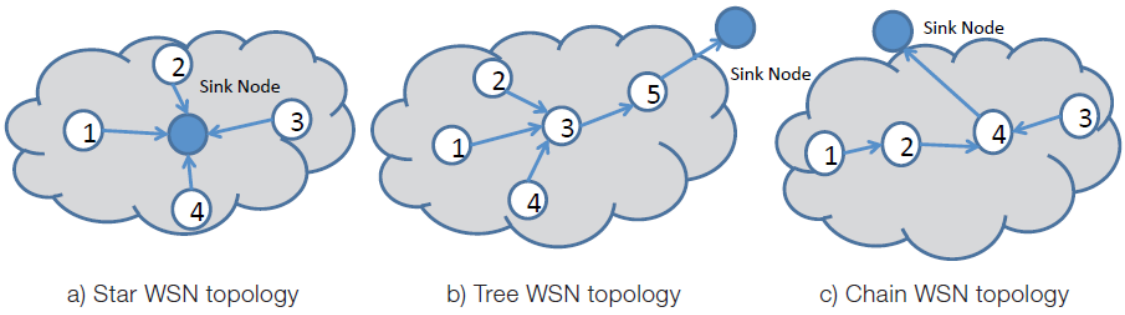


Figure 2.4: WSN topologies [2]

In the star topology, the nodes are placed at only one hop distance from the sink node so that redundant data can be collected from different sensors. The post-processing of information is done by the sink node. The tree topology supports multi-hop communication and the data is post-processed by sink node, they are typically connected to the internet to monitor the network managed by the client.

With respect to IoT, WSNs can be seen in two different ways:

1. Every node is a distinct entity or
2. The entire network is an entity which has full information about the network and can be accessed through the sink node. It will be more challenging since WSNs can be integrated into more complex networks.

2.2 Query Processing in Wireless Sensor Networks

Querying is a special way of aggregating the data and for any given query, there exist multiple query plans, representing alternatives for retrieving data for the query [39]. A query plan can be divided into two components for a simple aggregate query. Every query requires the data from spatially distributed sensors and must deliver these data from the nodes to a destination node for aggregation by providing appropriate communication technology within the network which is the query plans communication component [40]. The leader of the aggregation is the destination node. Furthermore, The computation component of a query plan computes both the aggregation at the destination node and already partial aggregates at intermediate nodes as well [40].

The energy of the nodes is very important when applying query processing strategies for sensor networks. If the communication and computation are coordinating well, which is possible to compute partial aggregates at intermediate nodes. Hence, which reduces the amount of the data needed to be sent, as a result, saves energy. There are two approaches and query systems for aggregating the query.

2.2.1 Query Approaches

The types of query approaches are [3]

1. Centralized approach
2. Distributed approach

2.2.1.1 Centralized approach

The centralized approach is shown in Figure 2.5.

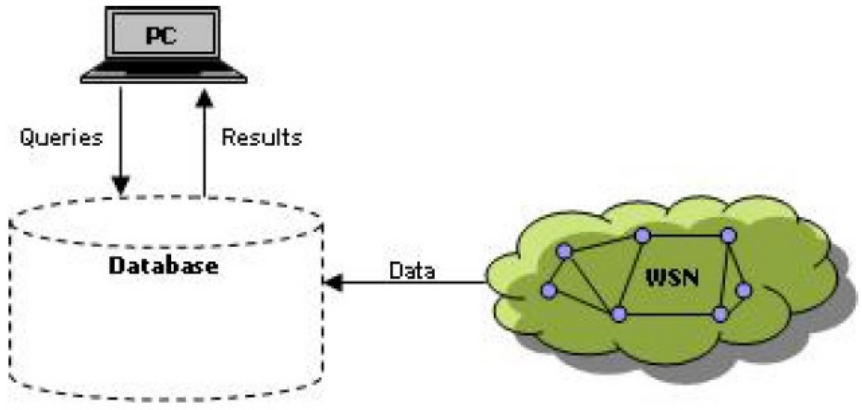


Figure 2.5: Centralized approach [3]

The Centralized approach is a traditional database approach, where data is extracted from the sensor nodes and is stored in the database. Then the client queries the database [3].

However, this approach is not suitable for query processing in WSNs due to a couple of reasons which are the data collected by the sensor nodes might be irrelevant to the query or application and sending a large amount of raw data to the centralized database, sensor nodes are powered by the batteries with restricted capacity. Therefore, sending all the data consumes a lot of energy. Therefore, sensor networks must save energy to extend their lifetime [41].

2.2.1.2 Distributed approach

To store larger volumes of data than the data stored in embedded and sensor networks, distributed approach or Database-oriented data aggregation approach was introduced. This approach is shown in Figure 2.6.

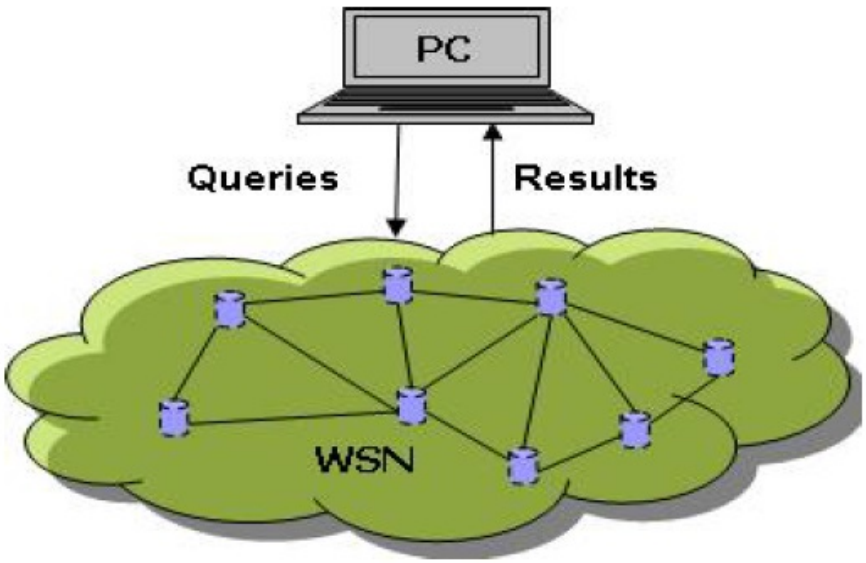


Figure 2.6: Distributed approach [3]

2.2.2 Query Processing Architecture

The query processing approach improves the energy efficiency dramatically with the typical measure of performance in sensor networks of data collection applications [4].

Sensor networks provide challenging computing and programming environment, where the devices are very small and since the software on them is not really complex, they seldom crash due to the insufficient energy, and the OS running on them provides no benefits to reduce such failures [4]. Generally, debugging is done using few LEDs on the device, while sharing information and processing, programs must carefully manage energy as they are highly distributed.

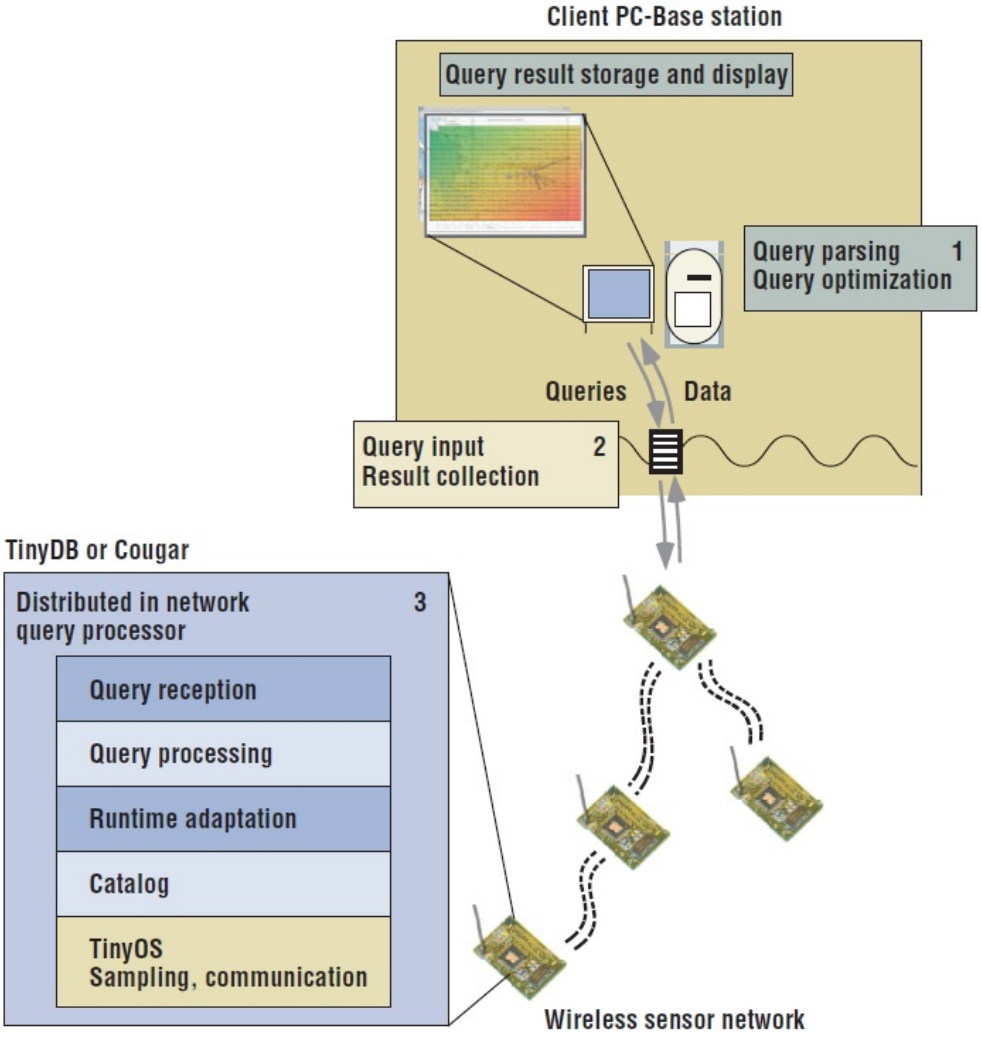


Figure 2.7: Architecture for Query processing in Sensor Networks [4]

Due to the restrictions in a computing environment, data collection systems must carefully manage resources such as energy and must be aware of and manage the nature of sensor networks. To provide logging facilities, storage for the offline analysis, systems should reduce and summarize data online.

The architecture has two main parts and described below [4]:

1. Server-side software
2. Sensor-side software

Server-side software

Running on the PC of the user (base station) where the software passes queries in the network, then delivers them to concerned nodes in the network and collects results from them.

Sensor-side software

Running on the sensors where the software is distributed in-network query processor, which is shown in detail in Figure 2.7. One of the sensors is the sink node, which communicates with the base station. Query processing steps on the sensor side software is shown in Figure 2.7.

2.2.2.1 Introducing Queries and Query Optimization

The client gives the query, which is in turn queried at the server level in a simple query language like SQL. This explains how the data can be collected and how it will be combined and aggregated. Introduction to the query systems is shown below.

Query Systems

Database-oriented systems for sensor networks are TinyDB [42], COUGAR and PLANetary [18] [43].

TinyDB

TinyDB has been developed at University of California, Berkeley [42]. TinyDB runs on TinyOS, which is an OS for sensor nodes based on the C dialect nesC [18]. Furthermore, it is platform dependent since an extended toolchain is required and its development ceased at version 0.2 in 2003 [18].

TinyDB is a simple query system which provides Structured Query Language (SQL) interface and allows the client to query the network using declarative queries and

collects data from sensor nodes, filters and aggregates it together [41]. The aggregated data send to the Data sink or Base station via an energy-efficient in-network processing algorithm [3].

TinyDB mainly focuses on

- In-network aggregation: computes aggregations within the network)
- Query language: simple SQL queries
- In-network query processing: In TinyDB, Query has to be delivered to only concerned nodes for the query execution.
- Multiple queries: TinyDB supports multiple queries to be run on the same set of network nodes at the same time [3].

COUGAR

COUGAR supports x86-architecture sensor nodes only. Furthermore, the last activity regarding COUGAR is dated 2005, and the project currently seems to be on halt and code which has to be run by the sensor nodes (QueryProxy) has never been made publicly available [18].

In Cougar, instead of sending all the raw data to Data sink/Base station and the results should be aggregated at the intermediate nodes. The aggregated results at each intermediate node communicate towards until it reaches the base station.

TinyDB and Cougar support grouped aggregation queries. Aggregation reduces the quantity of data that must be transmitted through the network. Therefore, it can reduce energy consumption and bandwidth, and it replaces with more expensive communication operations with cheaper computation operations [40]. Moreover, it is extending the sensor network's lifetime significantly [44]. TinyDB and Cougar support non-nested and conjunctional links between the statements [18].

Alternatively, the user can set queries to run for a specific time interval via EPOCH duration or FOR clause.

PLANetary

PLANetary has been developed at the Professorship for Computer Engineering at Technische Universität Chemnitz [45]. It is a database-oriented data aggregation system, light-weight, platform independent (not depending on specific OS) and focuses on energy efficiency [18]. With using the PLANetary library, the user allows sending queries to a sensor network which are then answered by this network. The functional core of PLANetary works similar to TinyDB or COUGAR. Because of

the flexible mechanisms, data querying and data preprocessing become easy.

PLANetary supports in finding energy-efficient routes through the sensor network and does not apply a single routing strategy.

Query models

Like SQL, queries in Cougar and TinyDB consist of SELECT, FROM, WHERE, GROUP BY, HAVING blocks to support selection, join, projection, aggregation, and grouping [4].

Clients store different kinds of sensor readings, and the system sends the results to the client. The time span result produced an EPOCH, and the EPOCH duration is time taken between the results.

Figure 2.8 shows query template in TinyDB.

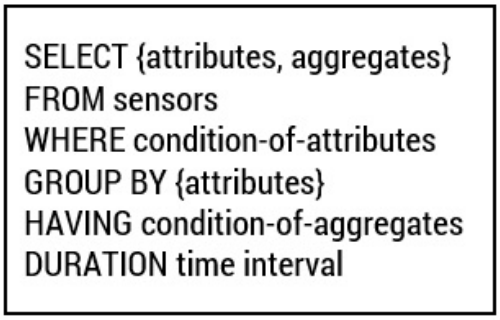


Figure 2.8: Query template in TinyDB [5]

Query example

Figure 2.9 shows an example of a query in TinyDB.

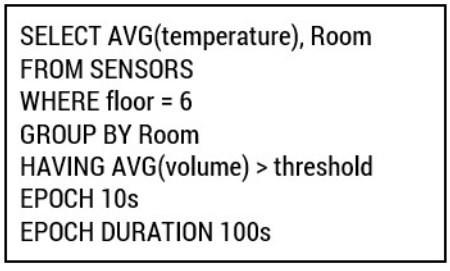


Figure 2.9: Query example in TinyDB [5]

In traditional database systems, queries define a logical set of data that the user is interested in or that is required for the user. Nevertheless, it does not describe the software modules, operators or algorithms. The system uses to collect the result set, and it can choose the operator and any plan for any logical query. This type of approach is called declarative approach.

For instance, to find the average temperature of the fourth-floor sensors, the system may collect readings from all the sensors, then filter for fourth-floor sensors and then compute the average, otherwise, it may request the sensors on the fourth-floor only to provide their temperatures and then take its average [4]. The second plan is the best choice because it requires only sensors on the fourth-floor to collect and give their temperature readings in a sensor network.

The process of choosing the best possible plan to execute the query is called query optimization. Query optimizers work on a set of possible plans, assigning a cost to each plan based on estimated costs of each of the operators, and then choose the lowest cost plan [4].

2.2.2.2 Query dissemination and Result collection

The system disseminates the query into the network after query optimization process.A routing tree is a communication primitive established at either the at the sink node or at the base station, and it is formed as nodes and forwards the query to other neighbor nodes in the network [4]. The parent of the network transmits the query, and all child/leaf nodes receive the query and process it and push it to their children, who in turn forward it to their respective children until the query reaches the whole network.

Nodes communicate with a wireless transceiver which is referred as radio transmission. Each radio transmission message consists of a level or a hop count representing the distance from the broadcaster to the sink node or the base station. Therefore, this can be determined and calculated own level by node selects a parent node which will be the responsible for forwarding the query results of the nodes and their children nodes to the base station [4]. The network can have multiple routing trees and supports multiple queries at a time with different base stations when the nodes have multiple parents. This type of communication topology is called tree-based routing.

The Figure 2.10 is an example of routing tree and sensor network topology. Thick arrows represent parent nodes, and dotted lines represent nodes that could communicate with each other, however, do not have any route between them.

A node can select a parent node from some possible nodes; the simple approach is to choose the ancestor or predecessor node at the lowest level [4]. Practically, choosing the proper parent node is very important regarding the efficiency of data collection

and communication.

The information is disseminated from the sensors to the base station. Each node has a connection to the base station which is away from few hops in the routing tree. The work of the base station is to collect the data from the sensors and forward the results via route. In TinyDB and Cougar, the routing tree evaluates over time, when new nodes come online, nodes run out of energy or interference patterns change [4].

In TinyDB, nodes maintain the tree by having a set of several parent nodes and measuring the quality of the communication link for all the parent nodes. The node takes a new parent, when the links quality to the present parent is worse than the quality of another nodes parent.

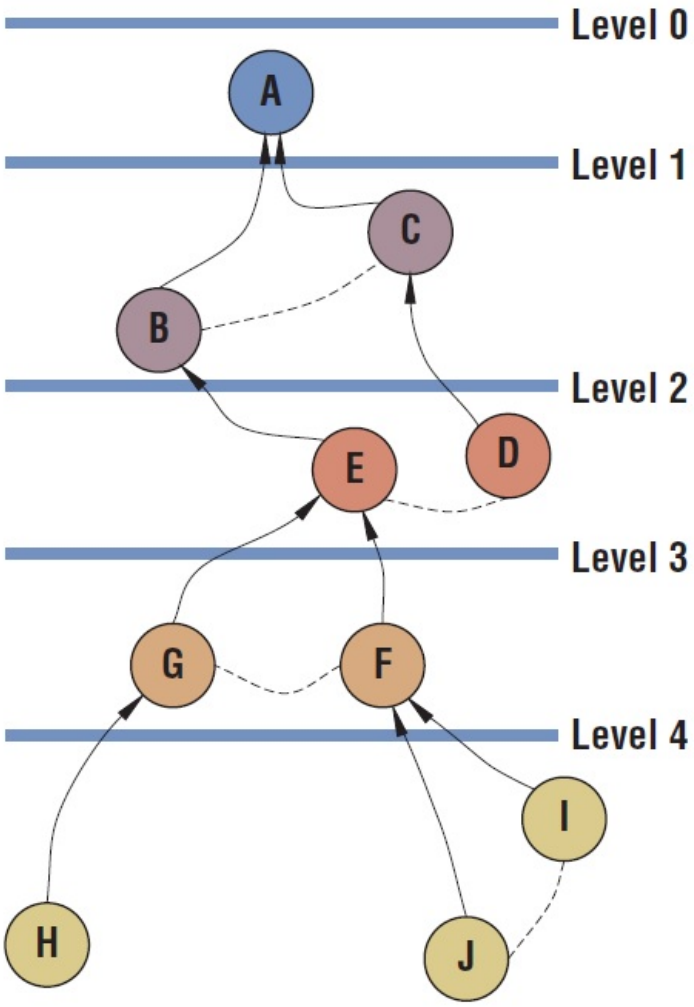


Figure 2.10: Sensor network topology and Routing tree [4]

2.2.2.3 Query processing

Each node begins to process the query, after query optimization and query dissemination. Processing is a simple loop: once per EPOCH in the EPOCH duration [4]. At each node, a special acquisition operator collects information from sensors corresponding to the fields mentioned in the query.

Using selected query plan in the query optimization phase, the query processor routes this set of sensor readings. The special acquisition operator in the query plan applied in static or fixed order. The data acquisition operator uses a list of available sensor readings to map names mentioned in queries to low-level operating system functions that can be invoked to provide their values [4]. By adding new sensors, the users extend the sensor network, and different software interfaces can be used to access the sensors. For instance, in the TinyDB, users can run queries using sensor attributes such as pressure, sound, temperature, light and also query attributes which reflect the OS or state of the device.

Below Table 2.1 describes some common query processing operators used in Sensor Network Query Processors(SNQPs)

Operator	Description
Data acquisition	Acquire a reading from a sensor, such as a Temperature sensor reading
SELECT	Ignores readings that don't satisfy a particular Boolean predicate. For instance, the predicate temp > 30°C. ignores readings under 30°C.
Aggregate	Combine or merge readings according to an aggregation function. For example, AVG(temp) computes the average temperature value over all nodes.
JOIN	Concatenate two sensor readings. AVG(temp) WHERE floor = 4, AVG(temp) WHERE floor = 6 AVG(temp) WHERE floor = 4 AND floor = 6.

Table 2.1: Sensor network query-processing operators [4]

The Figure 2.11 describes query processing for the simple aggregate query, "average temperature on the fourth floor once every five seconds" [4].

Here, the query plan consists of three operators:

- A data acquisition operator,
- A select operator that checks whether the value of the floor attribute equals to 4

- An aggregate operator that computes the average temperature from the local node and nodes ancestors that are on the fourth floor. Once per EPOCH, each sensor applies this plan, and the aggregated data produced at the sink node is the result of the query. The partial calculations of averages as SUM, COUNT pairs combined at each intermediate node in the query plan to calculate a running average as data goes up the tree [4].

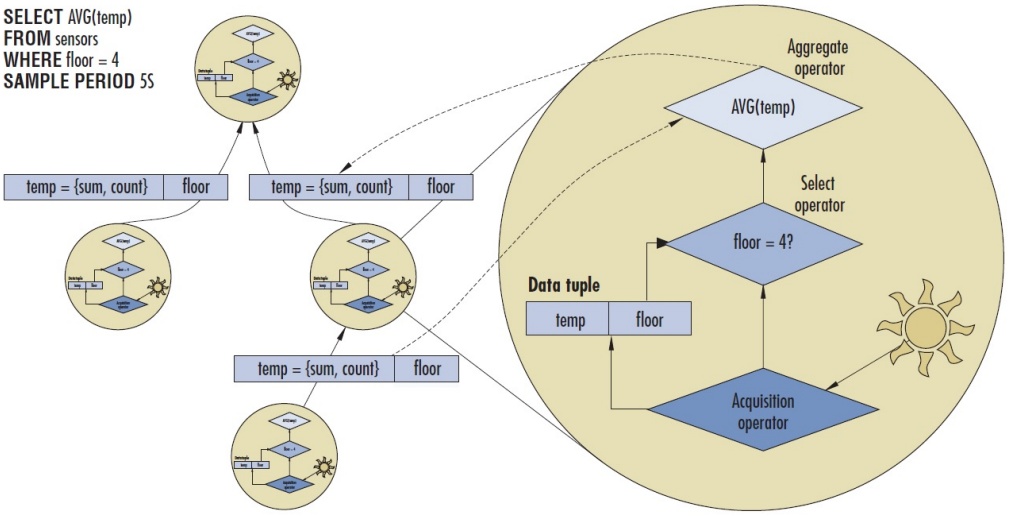


Figure 2.11: Execution of simple aggregate query [4]

Sensors must wait until they get acknowledgment from their child nodes before sending their averages of reading values, and average records must be represented and then can be combined as they travel up the tree (SUM and COUNT).

Finally, if the nodes are not moving or changing, the values of the floor will be constant meaning that nodes on other floors will not produce the values for the query [4].

2.3 Car2X Communication

Recent research indicates that Intelligent Transportation Systems (ITS) is a key technology for the Car2X communication applications such as improving traffic efficiency, road safety, and comfort driving [19]. Vehicular Ad Hoc Networks (VANETs) have a great potential to enable applications for traffic safety, sustainable mobility, and efficient transportation [19]. In the design and characterization of vehicular applications, the communication methods and routing algorithms play a vital role. More details on the communication technologies and routing methods will be given in later chapters.

It is a concept in the focus of research and development that Car2X communication technology enables data communication between the vehicles (V2V–Vehicle to Vehicle) and between the vehicles and road infrastructure (V2I–Vehicle to Infrastructure) and which also provides real-time information regarding traffic and weather conditions as well [23]. The Wireless communication technologies such as Wi-Fi, Zigbee, and Bluetooth are used to exchange information between the cars and with roadside units. Communication is possible in both directions (Car2Roadside and Roadside2Car). The Wi-Fi 802.11p standard is specially designed for radio transmission in Car2X applications especially Car2Car. The secure transmission of vehicle and traffic information with a small delay provided by using a direct connection between road users and infrastructure.

The Figure 2.12 shows Car2X communication technologies.

Figure 2.12: Car2X communication [6]

After testing, these cooperative systems are now on the approach of being deployed large-scale across Europe: Automotive companies, as well as road and infrastructure operators are currently setting up a Car2X corridor from Rotterdam via Frankfurt/M to Vienna [23].

Vehicle–2 –Vehicle (V2V)

Only vehicles communicate with each other and to send packet/information from the source vehicle to destination vehicle and can have any number of vehicles between them. It is not a continuous communication (Irregular intervals) [46].

Applications

Single-hop notification such as collision warning, lane changing and Adaptive Cruise Control (ACC). Whereas, Multi-hop notifications are accident alert and traffic monitoring.

The V2V and V2I communication is shown in Figure 2.13

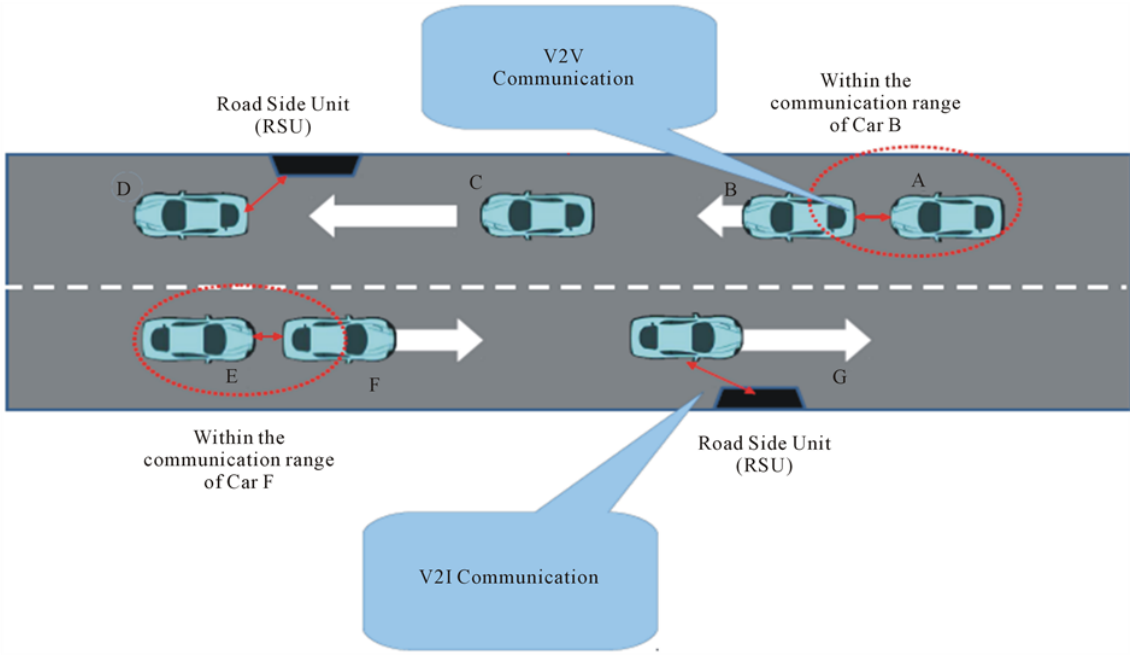


Figure 2.13: V2V and V2I communication [7]

Vehicle–2 –Infrastructure (V2I)

Vehicles communicate with Infrastructure and Vehicle will be source and Infrastructure will be the destination. It is also not a continuous communication (Irregular intervals) [46].

Applications

Infrastructure applications are not available yet. However, the Vehicle–2 –Roadside applications are traffic congestion, location-based services, high-speed tolling, accident alerting, and mobile infotainment.

2.3.1 Architecture Layers

According to the application, there are multiple ways to transmit or exchange the information in the Car2X or vehicular network. Three communication regimes are in existence: Single-hop, multi-hop position based and bidirectional and used for the specific communication [8]. In bidirectional regime, communication is possible in both directions. In single-hop and multi-hop regimes, communication is possible in one-way. However, compared to the multi-hop, single-hop is a bit faster. Figure 2.14 illustrates the Vehicular communication architecture. On top of the architecture, have Applications and Communication Regimes and then have the remaining layers: Routing, MAC, and PHY. The security layer is orthogonal to these layers [8].

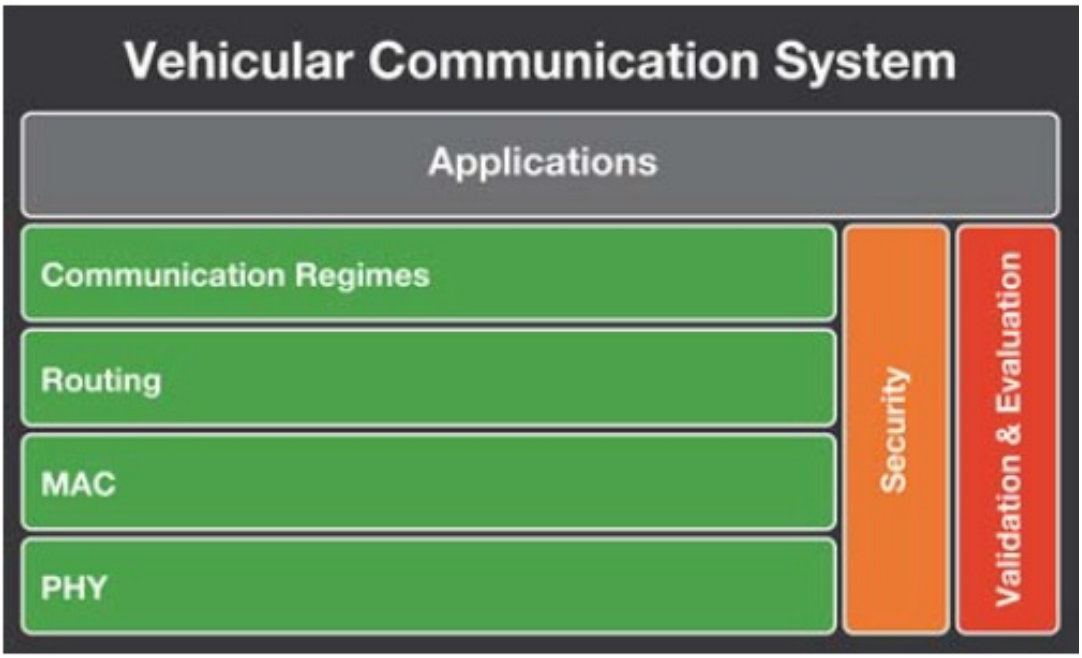


Figure 2.14: Communication Layers Architecture [8]

The MAC (Medium Access Control) layer is used to send the information of the vehicle to the application. It handles the transmission of the message and tries to avoid collision between messages.

The PHY (PHYsical layer) is a physical channel with dedicated frequency range used for transmission of messages in vehicular networks.

Routing layer is used to send data packets from source to the destination.

Security layer provides reliable security for the communication regimes.

Validation and evaluation will be having components of the systems and relationship between them, to the principles and with the environment [8].

2.3.2 Car2X Application Domain

The application domain of Car2X is shown in Figure 2.15.

- Safety
- Resource Efficiency
- Infotainment and Advanced Driver Assistance Services (ADAS)

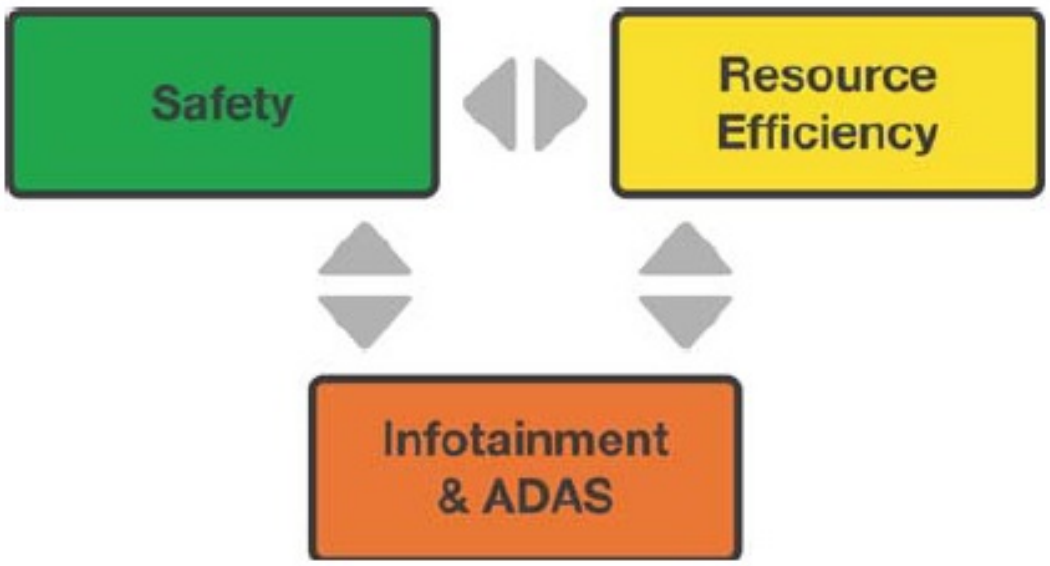


Figure 2.15: Car2X Application Domain [8]

Safety

Refers to an application, which increases the protection of people and vehicles which saves lives of people by avoiding accidents or minimizing the effects of an accident [8].

Resource Efficiency

Today, traffic congestion is one of the severe problems. Resource efficiency increases the traffic fluency, which is the most significant interest today as well as in the future. Better traffic efficiency results in less congestion and lowers fuel consumption which minimizes the economic and environmental impacts [8].

Infotainment and Advanced Driver Assistance Services (ADAS)

Provides information or entertainment to drivers and passengers, which makes driving more and more comfortable. Provides services such as toll payment, making phone calls, or listening to text messages, and playing music. It is implemented in navigation systems as well.

3 Concept Design

Due to the increasing number of nodes in distributed, embedded, and WSNs there is a need to retrieve local information and fuse it to global information efficiently [47]. The in-network processing is compulsory in local node storage because of the increase in data volumes.

The applications targeted by the future sensor networks, using traditional aggregation methods such as TinyDB and Cougar in embedded, WSNs will reach their limits [28]. These limits arise mostly with regards to data volume and energy efficiency. In traditional big data scenarios, relational databases must be succeeded by more suitable approaches [18]. However, where the traditional aggregation methods are not suitable for the problems well anymore, this approach is very useful in embedded, and WSNs.

There is an increase in data collected by a single node in future sensor networks, however, in most of the time, single values from the sensor network is not required extract by the client. Nevertheless, the demand for combination of values such as an average of temperature, maximum value, etc.

The focus on the aggregation of collective value sets of node subsets within a sensor network for immediate usage and there might be data collected by nodes which is irrelevant or not required to the current application [18]. To be able to filter the relevant data on the network allows reducing network traffic and storage requirements efficiently.

Therefore, most of the problems of big data in embedded, WSNs can be minimized using PLANetary. PLANetary is explained briefly in the chapter later and why it is suitable compared to other database-oriented data aggregation systems.

The main goal is to manage the large data volumes with regard to traffic-wise as well as energy efficiency.

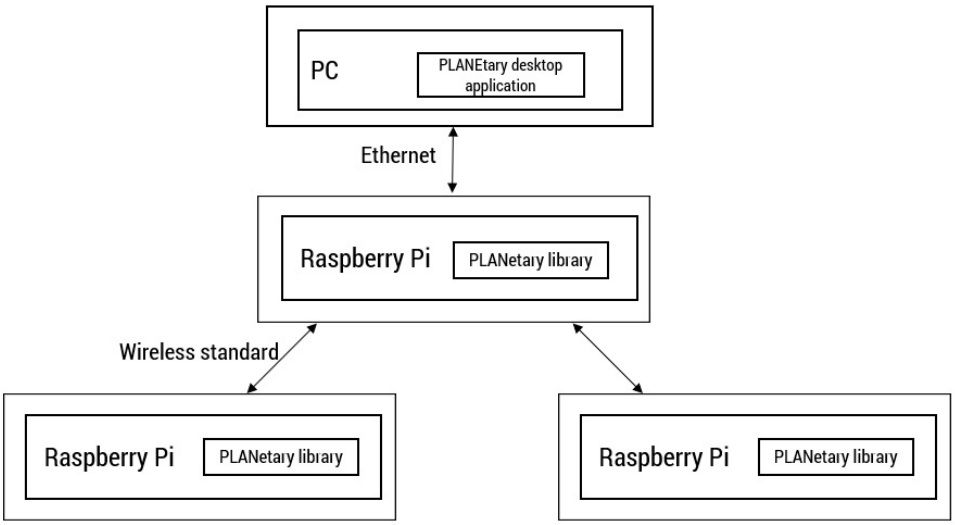


Figure 3.1: Concept overview

3.1 Wireless Communication Technologies

The wireless communication technologies are Wireless Fidelity (Wi-Fi), Bluetooth and ZigBee/XBee are today widely supported by all the mobile or embedded devices.

3.1.1 Bluetooth

Bluetooth offers a direct connection between devices which is short-range communication technology. The Bluetooth network topology is 1: N, where one Master device can connect to one or many (N) Slave devices, which is called a Piconet [12]. The master device controls the network.

Bluetooth Classic and Bluetooth Low Energy

The Bluetooth technology can be categorized into two: Bluetooth Classic and Bluetooth Low Energy (BLE) [12]. These two are not compatible with each other. However, both can be implemented by a device. These two have different range, power consumption, and throughput. Bluetooth Classic is mainly designed and used for streaming of data continuously during a connection; however, the connection is still maintained, even though there is no data needed to be transmitted [12]. Because of that Bluetooth Classic is not suitable for some use cases which require smaller amounts of data only.

The main advantage of BLE is that a small Bluetooth device can be powered with a tiny coin cell battery for a longer period due to the low power consumption.

Table 3.1 shows a comparison between the range, average power consumption and throughput between Bluetooth Classic and BLE.

Wireless Parameter	Bluetooth Classic	BLE
Range	< 10 m (Class II)	< 100 m
Power consumption	37.7 mA	0.024 mA
Throughput	Up to 2.1 Mbit/s	Up to 1 Mbit/s

Table 3.1: Bluetooth Classic and Bluetooth Low Energy specification [12]

BLE is ideal to maintain a Bluetooth connection for a longer period. Bluetooth Classic is more efficient for constant streaming of data, due to the higher throughput.

Bluetooth network is shown in Figure 3.2.

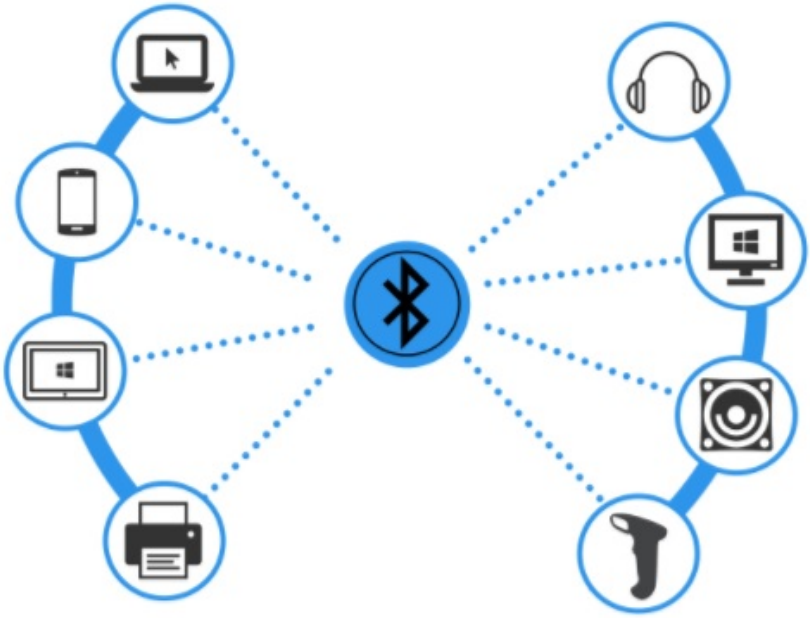


Figure 3.2: Bluetooth Network [9]

3.1.2 ZigBee module

Introduction

ZigBee is a wireless communication standard for connecting sensors. ZigBee, a specification for communication in a Wireless Personal Area Network (WPAN), has been called the Internet of things [2] [48]. The ZigBee enabled device at one end can communicate with ZigBee enabled at the other end.

ZigBee is a global, open and packet based protocol designed to provide an easy to use architecture for reliable, low power and secure wireless networks [48]. ZigBee and IEEE 802.15.4 are low data rate wireless mesh networking standards [49]. Therefore, ZigBee is used in industrial control applications to minimize or eliminate the costly and damages of wiring. The network does not much care about the physical location of a sensor, because of that ZigBee can also be moved from one place or other.

Some of the ZigBee applications are home and office, industrial automation, medical monitoring, low-power sensors and HVAC control [48].

Working of ZigBee

In ZigBee network, devices communicate with one other using digital radios signals. ZigBee network contains several types of devices, such as Network Coordinator which is a device that sets up the network. Network Coordinator is aware of all the nodes within its network and manages both the transmitted/received information and information about each node within the network as well [48]. Therefore, the ZigBee network must have a Network Coordinator to manage the network. Another device class or type is Full Function Devices (FFD), which supports all the 802.15.4 functions. These can be network routers, network coordinators, or devices that interact or communicates with the physical world or the environment which is known as Reduced Function Device (RFD).

ZigBee supports star, mesh, and tree topologies [50]. Star topology is very useful in ZigBee network. In a star topology, several devices communicate via a single router node when they are located close to each other [48]. The router node communicates with the network coordinator in a larger mesh network. Mesh networking allows for redundancy in node links when one link breaks down; devices can find an alternative route to communicate with one another device [48].

ZigBee network is shown in Figure 3.3.

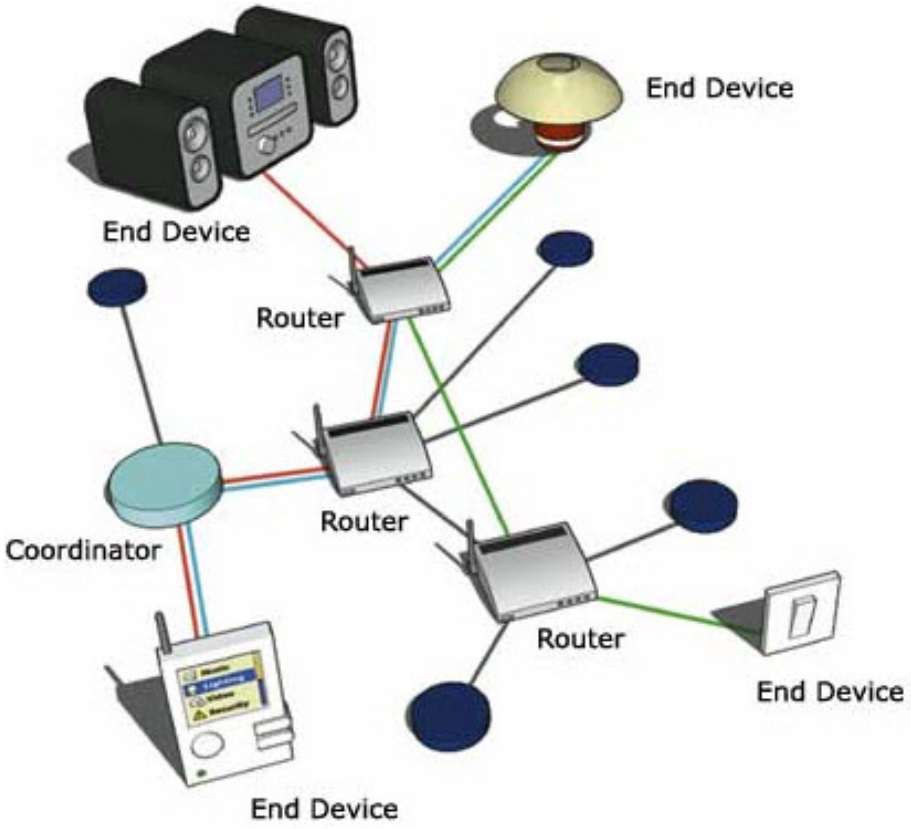


Figure 3.3: ZigBee network [10]

3.1.3 IEEE 802.11 WLAN

A WLAN (Wireless Local Area Network) is a wireless network of devices that communicate with radio frequency signals [12]. All the WLAN devices are Wi-Fi certified which are based on the IEEE 802.11 standard. Because of that, the term Wi-Fi is named after IEEE 802.11 WLAN. All devices within a WLAN can be divided into two categories, such as Access Points (AP) and Stations (STA).

Access Point (AP)

In WLAN Infrastructure mode, AP is the central device [12]. AP is responsible for transmission of data between the stations because all stations are connected to the AP. The stations can connect with other networks using an AP as a bridge for them. By acting as a bridge, the AP then needs to translate wireless IEEE 802.11 frames to wired Ethernet frames and vice versa [12]. The range of 802.11n APs is up to 300 m, and the speed 802.11n APs is up to 300-400 Mbit/s.

Station (STA)

In WLAN Infrastructure mode and Ad-hoc mode, the Station is a device, which supports IEEE 802.11.

Wi-Fi Network is shown in Figure 3.4.

Figure 3.4: Wi-Fi Network [11]

IEEE 802.11 WLAN has two basic operation modes: Infrastructure mode and Ad-hoc mode [12]. Today, most of the Wi-Fi networks are using the infrastructure mode.

Infrastructure mode

Infrastructure mode consists of one AP and multiple stations connected to it, and it is a star topology network [12]. The AP is needed to keep on broadcasting frames to announce its presence and maintain communication with each station even when there is no data to be transmitted. Depends on the throughput capability of the AP device, the power consumption keeps on changing. The AP functionality is more suitable for devices, where the device is connected to a power source regularly.

Ad-hoc mode

Ad hoc consists of only stations or clients, and it is a decentralized network. The network control is distributed between the stations because there is no such AP to

control and manage the network. All the stations store a routing table and the routes to a particular destination in the network. According to the routing table on each device, the data transmission can be done between two end-points are forwarded from station to station until it reaches the destination.

A station is connected and communicated with all the other stations using links, shown in Figure 3.5. The stations should handle a dynamic restructuring of the network due to the connection and disconnection of the link at any time. To find out whether the link or a station has been added or removed, the stations typically announce its presence and listen for announcement broadcasting frame from its neighboring stations [12]. The routing tables of affected stations are updated, according to the changes network structure.

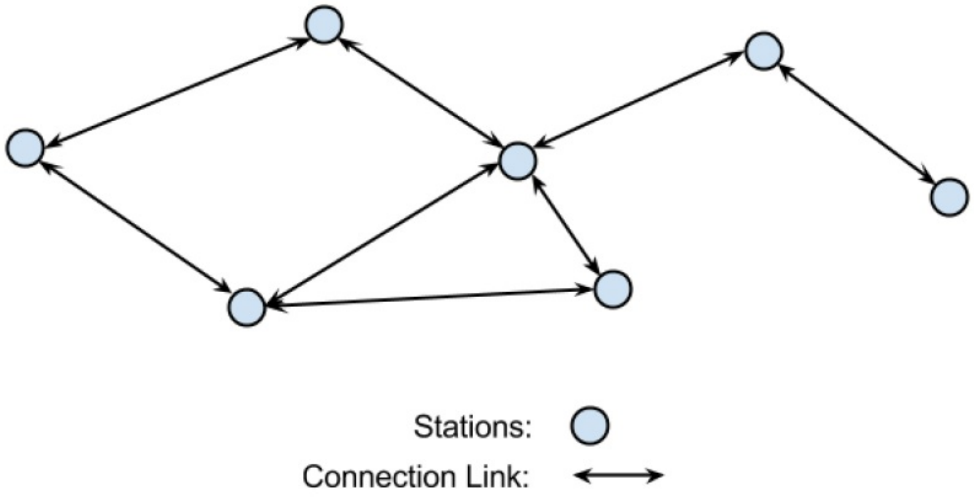


Figure 3.5: Ad-hoc network structure [12]

The main advantage of Ad-hoc mode, which is a peer-to-peer (P2P) solution that offers mobility and flexibility [12]. In Ad-hoc mode, the maximum data transfer rate of 11 Mbit/s, power consumption is 156 mA, and the throughput is lower compared to infrastructure mode.

The comparison of Wireless communication technologies is shown in Table 3.2.

Wireless Parameter	Bluetooth	Wi-Fi	ZigBee
Range	9 m	75 to 90 m	Indoors: up to 30 m Outdoors: up to 100 m
Power consumption	60 mA (Tx mode)	400 mA (Tx mode) 20 mA (Standby)	25 to 35 mA (Tx mode) 3 μA (Standby mode)
Raw data rate	1 Mbps	11 Mbps	250 Kbps
Frequency band	2.4 GHz	2.4 GHz	2.4 GHz
Physical/MAC layer	IEEE 802.15.1	IEEE 802.11b	IEEE 802.15.4
Protocol stack size	250 KB	1 MB	32 KB 4 KB (for limited function end devices)
Network join time	$>$ 3 sec	Variable, 1 sec typically	30 ms typically
Interference avoidance method	FHHS (Frequency Hopping Spread Spectrum)	DSSS(Direct Sequence Spread Spectrum)	DSSS(Direct Sequence Spread Spectrum)
Bandwidth	15 Hz (dynamic)	22 MHz (static)	3 MHz (static)
Nodes per network	7	32 per AP	64 K
Number of channels	19	13	16

Table 3.2: Comparison of Wireless Communication Technologies [21]

Where Wi-Fi Direct comes in.

3.1.4 Introduction to Wi-Fi Direct

Traditional Wi-Fi is a wireless technology based on IEEE 802.11 standards, developed for allowing multiple network client devices to connect with other network devices using a Wi-Fi Access Point (AP) [12]. AP handles the client to client communication.

Wi-Fi Direct technology is initially called Wi-Fi Peer-to-Peer (P2P), which is defined by the Wi-Fi Alliance (WFA) aimed at improving direct device to device communications in Wi-Fi enabled devices without using an Access Point (AP).The devices establish a P2P Group to communicate between Peer-to-Peer, as shown in Figure 3.6, even though Wi-Fi Direct devices do not belong to the same Wi-Fi network or without internet access. In a P2P Group, P2P devices must be capable of different roles such as P2P Group Owner (GO) or P2P Client. All the P2P devices are required to implement both P2P GO and P2P Client because of roles of the devices are dynamic.

P2P Group Owner: GO has the capabilities of AP that control a Wi-Fi P2P Group and enables connection to the P2P device [13].

P2P Client: A Wi-Fi Direct device which connects to a P2P GO [13]. In the P2P Group, A P2P Client communicates with the P2P GO [12].

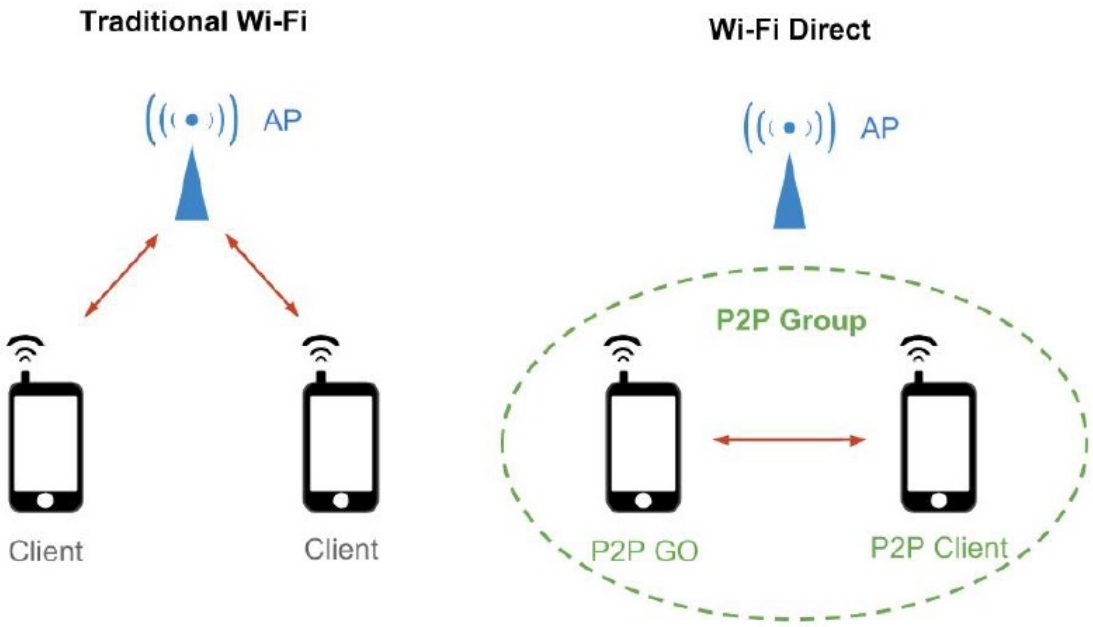


Figure 3.6: Traditional Wi-Fi network compared with Wi-Fi Direct [12]

Wi-Fi Direct specifications are shown in Table 3.3.

Wireless Parameter	Wi-Fi Direct
Range	200 m
Data Rates	Up to 250Mbps
Operating band	5/2.4 GHz
Channel bandwidth	20 MHz
Security	WPA2, AES 256 bit encryption
Coverage	Two way area
Equipment cost	No additional cost

Table 3.3: Wi-Fi Direct specifications [22]

Architecture of Wi-Fi Direct

Wi-Fi Direct devices, formally known as Peer-to-Peer (P2P) Devices, communicate by establishing P2P Groups [52]. The Wi-Fi Direct architecture consists of P2P GO, P2P Client and Legacy clients. P2P GO and P2P Client has explained above. Legacy clients may only function as a client and can also communicate with the P2P GO. P2P legacy devices neither belong to P2P Group nor support the improved functionalities defined in Wi-Fi Direct, However, they see GO as an AP [52]. Figure 3.7 shows 1:1 P2P Group and 1:n P2P Group.

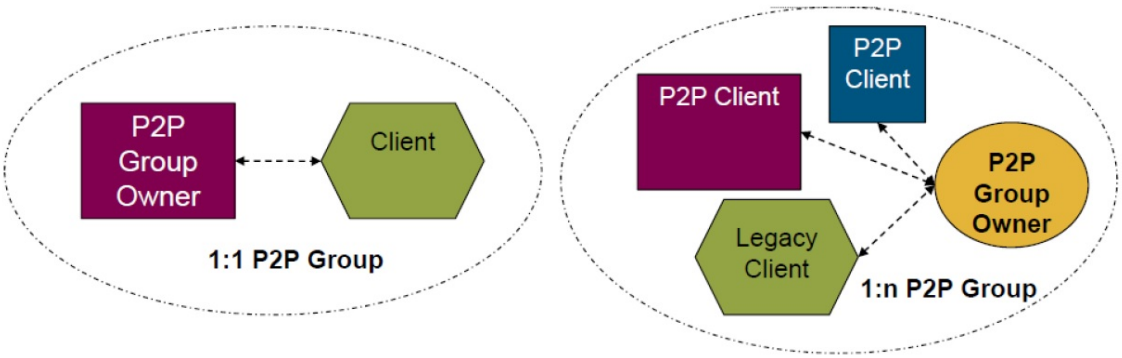


Figure 3.7: 1:1 P2P Group and 1:N P2P Group [13]

Finally, Wi-Fi Direct does not allow exchanging the role of P2P GO in the P2P Group [14]. If the P2P GO leaves the P2P Group, then the group is broken down, and it has to reinitiate the establishment of P2P Group [52].

Group Formation

P2P Group can be established using several ways, such as Standard, Autonomous, and Persistent formation [14].

In all three formation procedures, Wi-Fi Direct devices usually start by performing a traditional Wi-Fi scan phase, however, differs in later phases as shown in Table 3.4.

Standard	Autonomous	Persistent
1. Discovery • Scan Phase • Find Phase	1. Discovery • Scan Phase	1. Discovery • Scan Phase
2. GO Negotiation		2. Invitation
3. WPS Provisioning • Phase 1 • Phase 2	2. WPS Provisioning • Phase 1 • Phase 2	3. WPS Provisioning • Phase 2
4. Operational Phase	3. Operational Phase	4. Operational Phase

Table 3.4: P2P Discovery and Group Formation Procedures [12]

Standard Formation

The standard formation is used when P2P devices have to discover each other and then establish a P2P Group [14]. After the establishment of P2P Group, devices need to negotiate who will act as the role of P2P GO [12]. The Standard Formation is shown in Figure 3.8.

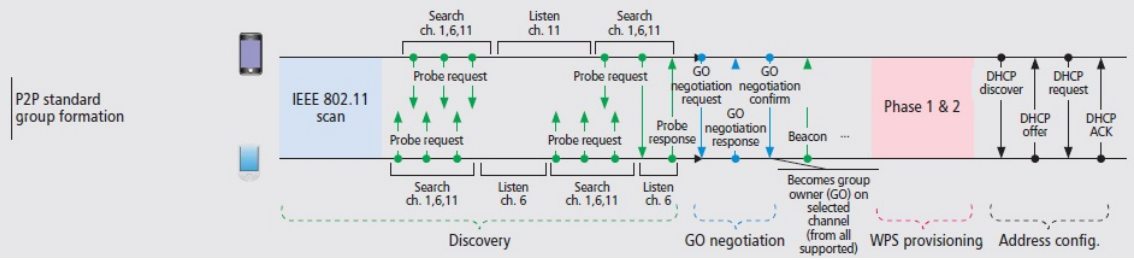


Figure 3.8: Standard Formation [14]

Furthermore, after the Wi-Fi active scan, the discovery phase also includes a find phase, to detect other devices Probe Requests and Probe Responses are used.

The GO negotiation phase starts after the P2P devices discovered each other.

The WPS Provisioning phase starts after the devices have discovered each other and agreed to the roles of P2P GO and P2P Client [12]. The Wi-Fi Protected Setup is

used to establish a secure communication [52].

Finally, set up the IP address configuration [14].

Autonomous Formation

The Autonomous formation is used when a P2P Device may create a P2P Group autonomously and becomes the P2P GO immediately [14]. Other P2P devices can discover the already formed P2P group using scanning mechanisms such as Wi-Fi active scan and then join as P2P Clients. Therefore, proceed with the WPS Provisioning phase and then Address Configuration phase [52]. The device who establishing the group does not need to go through the find phase, Because of that Discovery phase is simplified [12]. Go Negotiation phase is not needed [14]. The Autonomous Formation is shown in Figure 3.9.

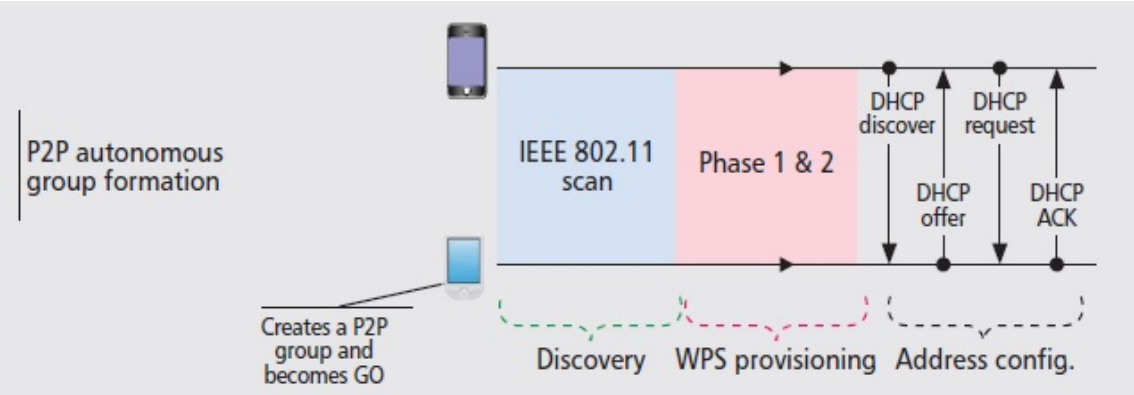


Figure 3.9: Autonomous Formation [14]

Persistent Formation

By using a flag during the formation process, P2P devices can declare a group as persistent. The network credentials are stored by the devices which are formed the group and assigned roles of P2P GO and Client to the devices in the P2P group [52]. Later, if the devices would like to connect, one of the device sends an invitation request to re-establish the persistent group with their original roles [12]. The Persistent Formation is shown in Figure 3.10.

Even though the devices which were part of a persistent group previously, the persistent group does not remember the invitation requests from the devices. Because of that the invitation fails, and group formation will continue as a standard formation from the GO Negotiation phase [12].

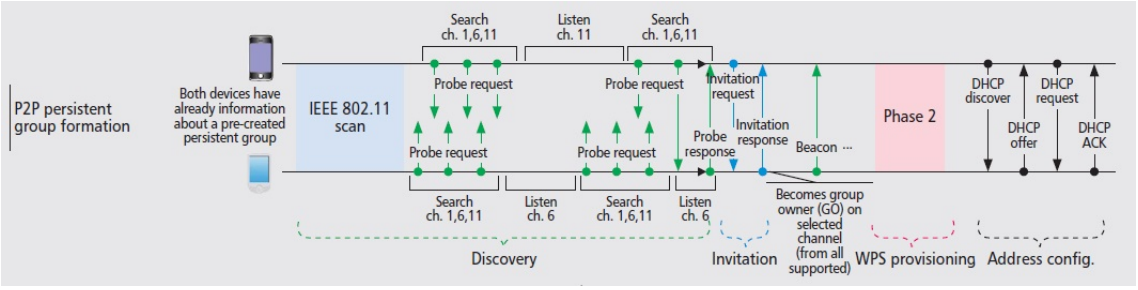


Figure 3.10: Persistent Formation [14]

Device Discovery

Device Discovery is to discover Wi-Fi Direct devices and establishing a connection between them [53]. The connection is established when one device sends a probe request while another device is listening and vice versa as shown in Figure 3.11.

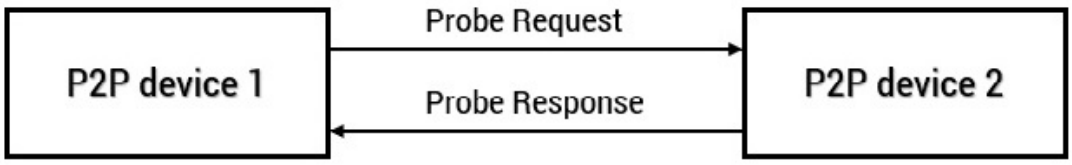


Figure 3.11: Device Discovery [15]

Scan Phase

Using traditional Wi-Fi scan through all supported channels (1, 6 and 11) starts discovery phase to find the neighboring P2P Groups and legacy Clients. The best potential channel is found by the P2P device for the later establishment of a P2P Group by using Wi-Fi active scan. The P2P device shall not reply to any Probe Request during this phase [12].

Find Phase

The P2P device starts the Find Phase after the Scan phase ends. The focus of scan phase is to confirm that two P2P devices in Device Discovery are in the same channel (1 or 6 or 11) to exchange device information and determine if a connection should be attempted or not [12].

The Wi-Fi Direct scanning channels are 1, 6 and 11 in the 2.4 GHz band, shown in Figure 3.12.

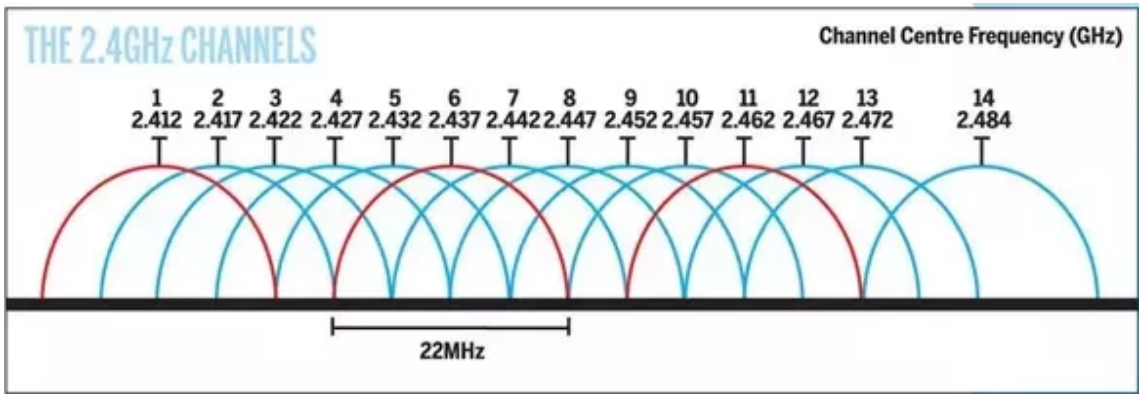


Figure 3.12: Channels in the 2.4 GHz band [16]

Two P2P Devices in Device Discovery

A P2P device in the Scan Phase may discover a P2P device in the Listen State [17]. The Find Phase is responsible for the two P2P devices to make sure they both are in Device Discovery phase and arrive on a same channel (1, 6 or 11) to exchange information of the device.

If any of the P2P device wants to connect, it has to do one of the following:

- Initiate GO Negotiation: To form a new P2P Group.
- Send a P2P Invitation Request frame: It requests the previously formed Persistent P2P Group, where one of the P2P devices is the P2P GO.
- Send a P2P Invitation Request frame: It requests the target P2P device joins a P2P Group and in which P2P device is may be the P2P GO or a P2P Client.

Device Discovery procedure for P2P devices shown in Figure 3.13.

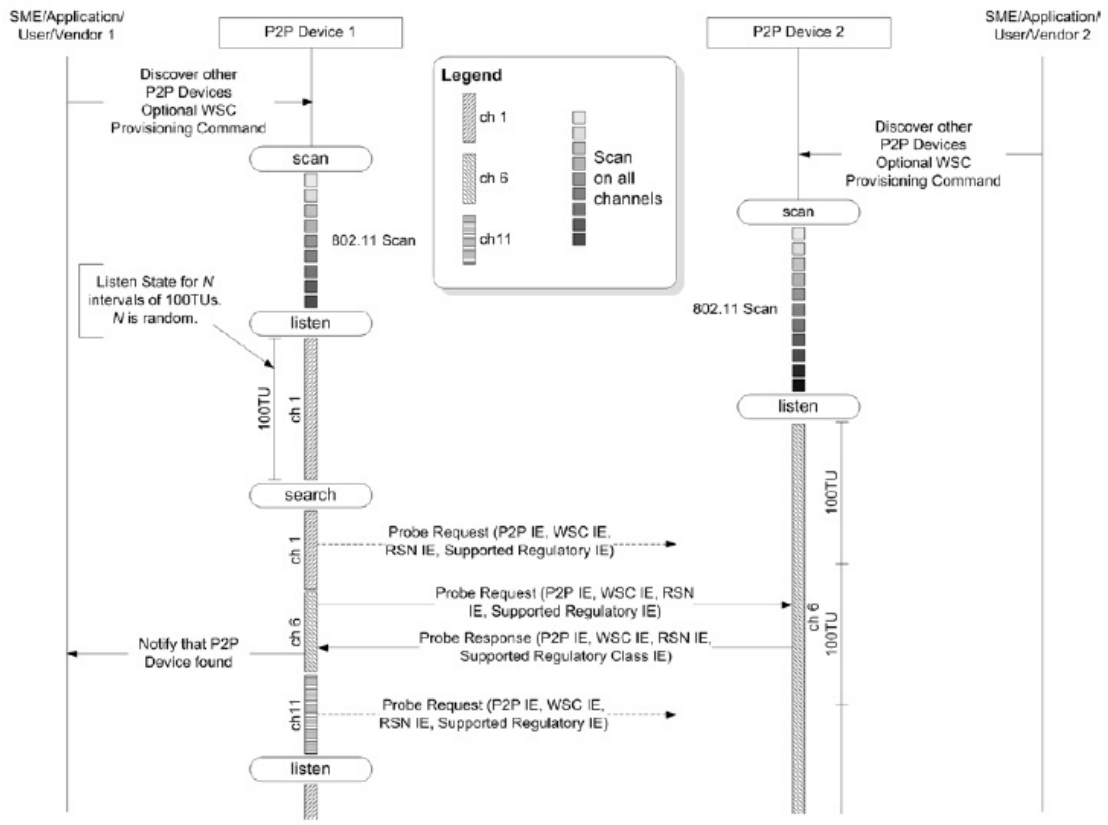


Figure 3.13: Device Discovery procedure for P2P devices [17]

Group Owner Negotiation

The GO negotiation phase starts after the P2P devices discovered each other. The role of the devices (P2P GO or P2P Client) depends on the GO Intent value (0 - 15) [52]. The device having the highest value becomes the P2P GO and lowest value becomes the P2P Client. If Go Intent value = 7, then there is a possibility for both devices to become GO.

For instance, X1-GO Intent Value of Raspberry Pi #1 and X2-Go Intent Value of Raspberry Pi #2. If X1 > X2, Raspberry Pi #1 will act as P2P Go and Raspberry Pi #2 will act as P2P Client else vice-versa.

The GO negotiation is shown in Figure 3.14

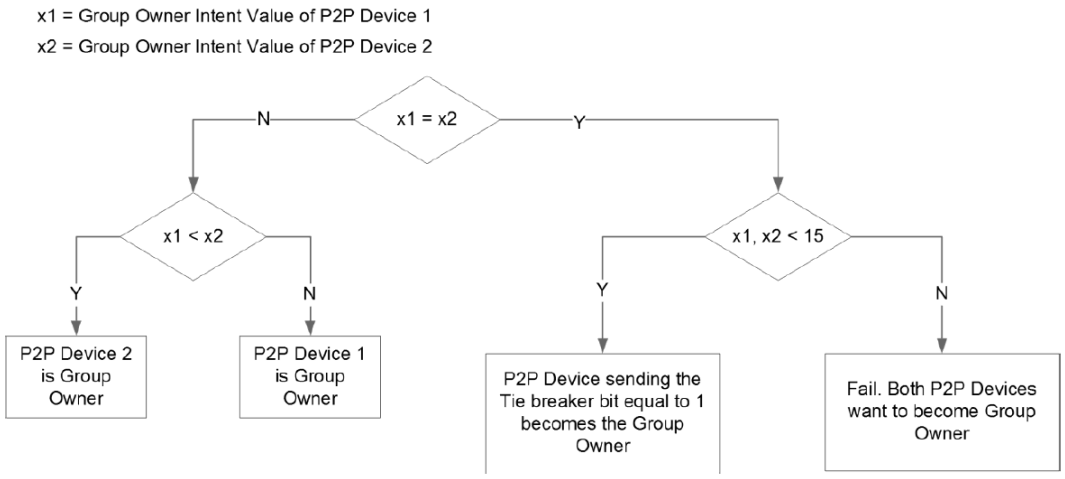


Figure 3.14: Group Owner negotiation flowchart [17]

WPS Provisioning

The authentication part starts after the GO negotiation and roles have been assigned to devices as P2P GO and P2P Client. The Wi-Fi Protected Setup (WPS) is used to set up and support the connection in Wi-Fi Direct devices. By adapting the WPS protocol, the P2P Group is required to implement two parts, Internal Registrar (P2P GO) and Enrollee (P2P Client) [12].

Phase 1

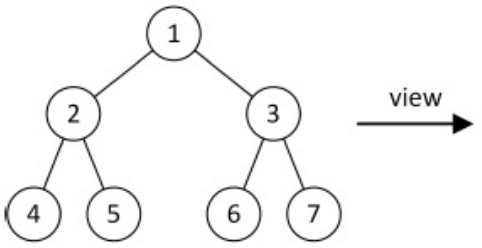
By exchanging frames (GO Negotiation/Response/Confirmation), the Internal registrar generates and issues the network credentials to the Enrollee.

Phase 2

Using the new authentication credentials, the Enrollee reconnects to the Internal Registrar.

3.2 PLANetary

Database-oriented data aggregation concept has been reintroduced [18] to handle data volumes which are larger than the information stored in present embedded, WSNs. The main concept is viewing or modeling the network as a virtual table in the database-oriented approaches in wireless sensor networks [28]. In the virtual table, the row represents sensor nodes of the network, where each node of the sensor network has sensors, and column represents each sensor of a node (called a node attribute). The nodes are not required to have the same sensors, so for some nodes, in specific columns, values may not exist. Node attributes may be simple sensor readings (such as average of temperatures and pressure etc.) or abstracted information (such as "traffic congestion detection" or "road ahead blocked warning" for C2C applications). When using static routing, nodes already know its parent node and child nodes established by a selected routing strategy. When using dynamic routing, during query execution time the route to take for a query may be determined [47]. Figure 3.15 shows an example ,where sensors are Temperature and Pressure.



Node-ID	Parent-ID	Temperature	Pressure
1	-	20°C	100 kPa
2	1	22°C	100,05 kPa
3	1	22°C	100 kPa
4	2	21°C	...
5	2	21°C	...
6	3	20,5°C	...
7	3	21°C	...

Figure 3.15: Sensor network as virtual table [18]

The issues produced by big data in embedded, wireless sensor networks resolved by the database-oriented approach. Existing systems such as TinyDB and Cougar lack several functionalities needed for future networks. Because of that PLANetary came to the existence.

Database-oriented approach consists of two phases [18], such as

1. Query propagation
2. Result aggregation

After the statement, the query has to be sent to all concerned nodes in the network. Furthermore, information at the nodes has to be aggregated and combined in-network and the results required to be sent back to the client [18]. If one of the below statement is true, the node executes the query:

- The conditions of the query are satisfied by the node and then the query executes
- The node sends the results from its children to its parent until they reach the base station or sink node

Query propagation and Result aggregation are shown in Figure 3.16.

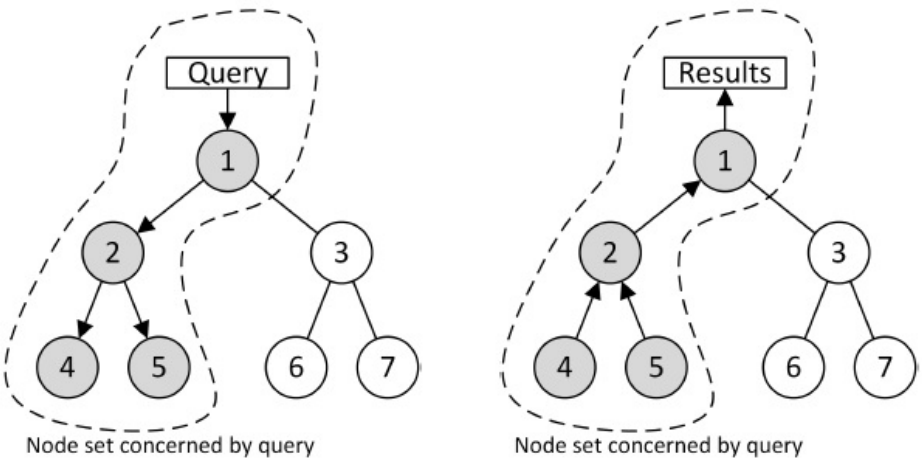


Figure 3.16: Query propagation and Result aggregation [18]

PLANetary is closer to the SQL dialect than COUGAR [28]. In TinyDB, SQL keywords are SELECT and FROM. In PLANetary, those keywords replaced with SENSE and AT to make a difference from TinyDB.

Compared to TinyDB and COUGAR, the major improvement of PLANetary is to support disjunctional (OR) and conjunctional (AND) links between nested conditions and condition statements as well [47]. Because in real applications, a nesting level of one is often required, etc. Therefore, PLANetary allows more complex queries and it is more comfortable for the SQL users with experience and the multiple queries in TinyDB can be formulated as a single query in PLANetaryQL [18]. It improves the time needed for query propagation, network traffic, and energy consumption.

A sample query is shown in Figures 3.17 and 3.18, where two queries in TinyDB can be formulated as single query in PLANetary.

In TinyDB

```
SELECT AVG(Temp)
FROM SENSORS
WHERE floor = 3

SELECT AVG(Temp)
FROM SENSORS
WHERE floor = 6
```

Figure 3.17: Two queries in TinyDB [18]

In PLANetary

```
SENSE AVG(Temp)
AT SENSORS
WHERE floor = 3
OR floor = 6
```

Figure 3.18: One query in PLANetary [18]

An example for more complex and nested query shown in Figure 3.19

```
SENSE AVG(Temp)
AT SENSORS
WHERE (ID > 3 AND ID < 9)
OR(floor = 3)
GROUP BY floor
```

Figure 3.19: Complex condition in PLANetaryQL [18]

The aggregation phase starts for the node starts after the query propagation, where the query has reached a leaf node of the query tree. Therefore, results are collected and fused in-network and aggregate functions (such as AVG, SUM, MAX etc.) are applied [18]. Consequently, from the leaf nodes, only the essential values are sent back to a parent node of the subtree. As a result, it reduces the traffic in the

network depending on the query is given by the user and for the values requested. The optimizations are very vital concerning the efficient use of energy of the overall system. The less power should consumed by the nodes which are not required to answer any of the running queries or forward query results [18].

If some parts of conditions may be unsatisfiable in subtrees, we may drop them [47]. Figure 3.20 shows an example. This shows the nodes of a subtree, where some conditions are dropped, as they can never be satisfied by the nodes of this subtree. Therefore, energy and time can be saved.

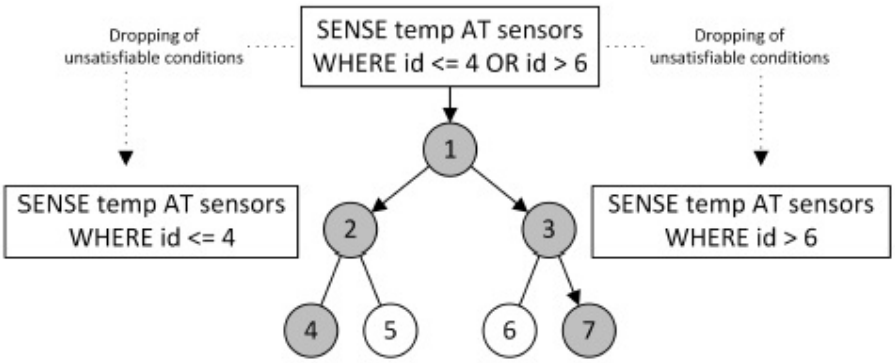


Figure 3.20: Dropping unsatisfiable conditions [18]

Contrary to TinyDB, PLANetary allows to define an arbitrary number of other virtual tables and the table definition consists of a table name and a predefined condition set [18].

There are two main advantages of virtual tables [18].

Firstly, query text becomes shorter when the conditions limit the query to a particular node are indicated by the virtual table [47]. However, after adding more conditions, the query might be changed on a virtual table.

Secondly, predefined query sets can support the study and routing process by combining nodes which belong to the definitions of the same virtual table [28].

3.3 Car2X Communication Scenario

This section gives a brief explanation about Car2X scenario and how it can be applied using PLANetary.

The number of vehicles on the road is increasing day by day which results in more congested roads and more accidents, etc. These congested roads are increasing noise and air pollution, driving is becoming more and more stressful. Table 3.5 illustrates statistics of a German driver. German drivers spend 35 hours/year stuck in road congestion, consuming around 11 billion liters of additional fuel [23]. In 2013, 7.4 billion € of the economic damage caused by traffic congestion in Europe. To avoid these situations as well as to handle them better, vehicles should get road information in a detailed way. The new concept Car2X communication has been introduced to solve these difficulties [23] and Intelligent Transportation Systems (ITS) is an investigation to reduce their negative effects of traffic congestion.

VANET (Vehicular Ad-hoc NETwork) [54] is a network where each vehicle on the street is a sensor node. When ITS (Intelligent Traffic Systems) sends the query to the network. As a result, the cars communicate with each other via wireless communication standard and then sends the result back. Cars will not send their information on their own. For instance, the middle of the road is congested, so the cars which are part of the congestion may report `congested` to neighboring cars [18]. The cars, which have already left conestion area and those which have not reached the congestion yet, do report `not congested` [18]. However, even though nodes change over time as they move, the street remains congested in the global perception. There is a need to retrieve local information and fuse it to global information as efficiently as possible with increasing node numbers in embedded, WSNs [47]. WSNs technologies can be applied to Car2X communication to make this data of the cars and roads available to ITS [8].

Hours stuck in congested road	35 hours/year
Fuel consumption	11 billion liters
Economic damage in Europe	7.4 billion €

Table 3.5: Statistics of German Drivers [23]

3.3.1 Vehicular Ad-hoc Networks (VANET)

Definition

Vehicular Ad Hoc Networks is an extension of Mobile Ad Hoc Networks (MANETs) [55]. The wireless technology runs on VANET. Sensors are used for the monitoring

of the environment and the data acquisition in WSN known as Vehicular Sensor Network (VSN) [19]. The VSN senses the data of traffic and environmental conditions (such as temperature, vibration, pollution, pressure, etc.), the information is processed by vehicular applications to generate messages and send the data over the network using wireless communication channels [56].

Characteristics

The characteristics of VANET are

Variant topology: The network topology is highly dynamic, because of the high speed and movement of vehicles.

Non-infrastructure network: V2V communications are based on the Ad-hoc network architecture [19]. To manage other nodes, there is no central or parent node. All the connections and transmission of data between the nodes are self-managed and self-organized.

Frequently disconnected network: The connection between the vehicles can be lost easily due to dynamic behavior of the network, which leads to packet loss in transmissions.

Battery power and storage: When the nodes are connected to the power source, there is almost no limit for the consumption of power.

Radio-communication aspects: Radio-communication is complex due to several factors, which are, not favorable conditions for propagation of the signal, regular interruptions of the radio-link, and radio frequency devices interference [19].

WAVE Family standards

The Intelligent Transportation Society of America (ITSA) suggested that the implementation of a standard for the physical layer and MAC (Medium Access Control) layer for VANETs, which is known as WAVE (Wireless Access in the Vehicular Environment) [19]. This standards family includes IEEE 802.11p and IEEE 1609 [57]. Figure 3.21 shows the architecture of WAVE.

In vehicular communication, the characteristics requirements of the physical and MAC layer are defined by IEEE 802.11p [57]. Orthogonal Frequency Division Multiplexing (OFDM) transmission for the physical layer (PHY) IEEE 802.11p.Besides, the communication channel is accessed by the MAC layer, then the stations can share the wireless medium efficiently [57]. The IEEE 802.11p standard defines the use of Carrier Sense Multiple Access/Collision Avoidance (CSMA/CA) for V2X communication [19]. The MAC layer is also a part of the transmission and the channel access

time, congestion control, the probability of receiving packets, and prioritization of messages.

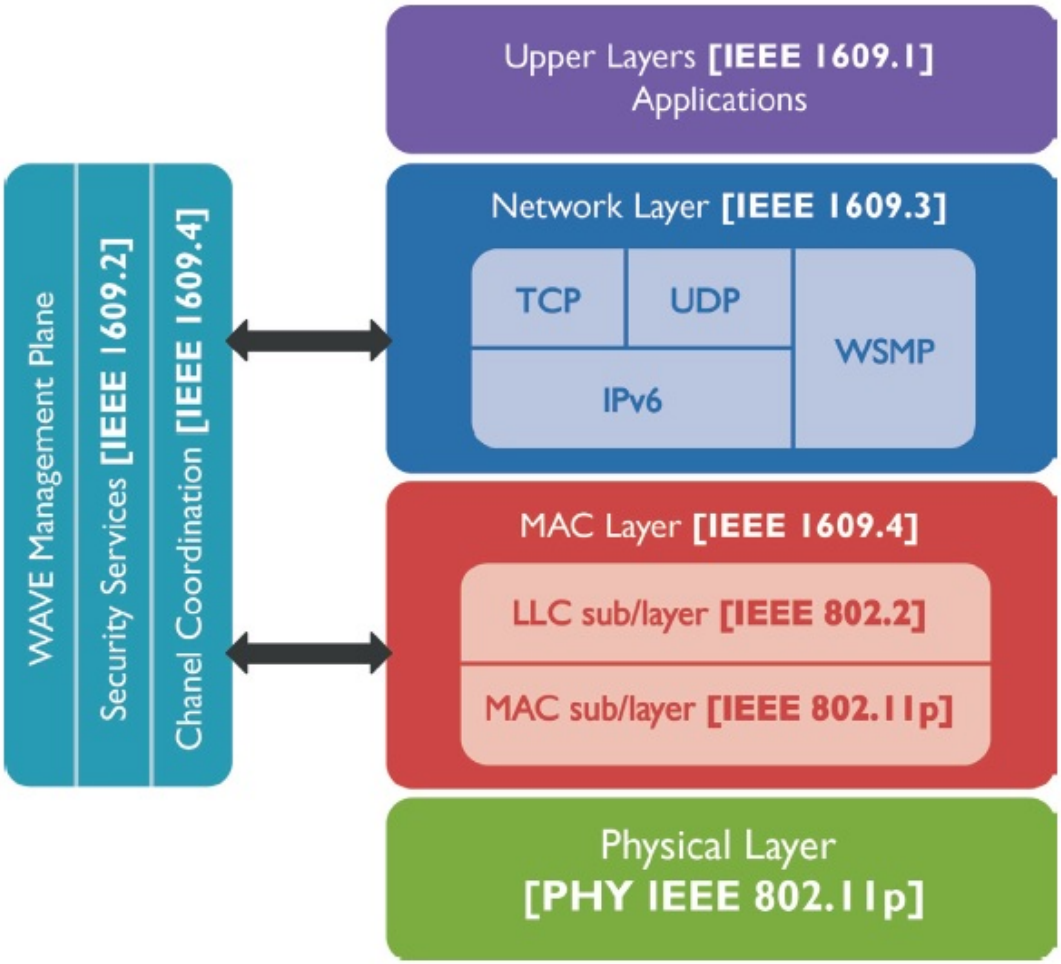


Figure 3.21: WAVE protocols architecture: IEEE 802.11p and IEEE 1609 [19]

The IEEE 1609 standards family defines the application and transport layer of WAVE architecture, management, and operation of the network and in the IEEE 1609.X standards define the description of the overall structure of the On-Board Unit (OBU), Road Side Units (RSU) and interfaces, communication models, switching of channels, multichannel operation, and security services for WAVE [19].

3.3.2 Routing Algorithm

Routing is used to send a packet from a given source to a destination [8]. To forward packets that contain information about the one vehicle to other vehicles can have any number of cars between them. This information can be about traffic information and danger warnings. If there is no vehicle within the communication range

a packet is stored and forwarded when the new vehicle comes into the range. In multi-hop routing, the information may be distributed in two ways: to the nodes in a specific area (geocast) or to all surrounding nodes (multicast) [8]. Nodes compute and use the best route to forward the messages to the neighboring nodes.

Routing protocols are classified into proactive, reactive and hybrid protocols [40]. Proactive routing protocols, such as DSDV [58], in advance, set the routes to all possible destination nodes..

Reactive routing protocols create and repair routes only on request of the node. Hybrid routing protocols, like ZRP [59], combine both properties of reactive and proactive routing protocols.

There are two classes of routing techniques

1. Static routing
2. Dynamic routing

Static routing

Each node already knows its parent node established by chosen routing strategy. In static routing, path uses to send packets is already known and initial configuration, maintenance is time-consuming. For the implementation purpose, one should have complete knowledge of the whole network.

Dynamic routing

The route to take for a query determined during query execution.

The on-demand routing protocols are Ad-Hoc On-Demand Distance Vector (AODV) Routing, Dynamic Source Routing (DSR) Protocol, Associativity Based Routing (ABR) and Temporally Ordered Routing Algorithm (TORA) etc [60].

A well-known routing algorithm designed for mobile Ad-hoc networks is the AODV [61] [62]. The properties of AODV are

- Scalability: supports large size of network with thousands of nodes [40].
- It prevents duplication or repetition of packets using destination sequence number and hop-count.
- It has already been implemented in several simulations and working perfect with regards to packet delivery ration, throughput [63].

Compared to other routing algorithms like DSR, DSDV [63]

- Higher packet delivery ratio
- Better end-to-end delay
- throughput is more

Therefore, AODV routing algorithm is suitable for the thesis.

Ad-Hoc on Demand Distance Vector (AODV)

AODV supports both multicast and unicast communication [8]. Routes are formed on-demand and maintained only if they are needed. In a tree structure, the management of multicast group members is being done [8]. The tree structure indicates the network topology needed to connect the multicast group members. Nodes represent network nodes and edges represents connections. Sequence numbers are utilized to guarantee that routes are up to date and updated properly.

The route construction in AODV, core mechanisms are the route request (RREQ), route reply (RREP) and route error (RERR) [8].

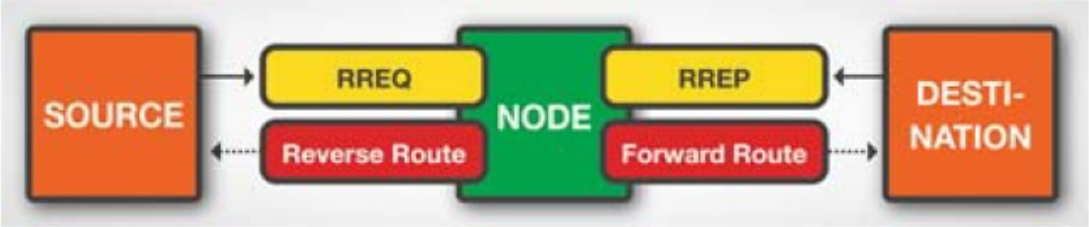


Figure 3.22: AODV route creation [8]

The source broadcasts an RREQ packet, before beginning transmission to an unknown node [8]. After receiving the RREQ, receivers update their routing table, and keep backward to find the source of the RREQ packet, called the Reverse Route.

With using RREQ ID and IP address of source node, RREQ can be recognized. It holds source sequence numbers and destination sequence numbers as well; the source sequence number is used for the backward reference in the routing tables of the receiving nodes, and the destination sequence number is the last sequence number the source received for a route to the destination [8].

An RREP is generated by the node when the node is a destination or the active route to the destination stored in the routing table with a destination sequence number greater than or equal to the destination sequence number in RREQ. When the

node does not generate an RREP, by using new and updated destination sequence number and the hop count, rebroadcast the RREQ.

When the RREP is broadcasting back through the nodes on the route to the source, in the routing table, created an entry for the route to the destination node (called as Forward Route) before broadcasting the RREP [8]. After receiving an RREP, using established a route to the destination, node starts transmission of data packets. RREP is used to update the routing table of the source if the incoming route is better such as having a lower hop count.

When the source stops broadcasting data packets, the links will be deleted from the routing table of intermediate nodes as they are time out. If the destinations are not found, a route error (RERR) is transmitted back to the source. Therefore, the broken link is detected [8].

To create multicast routes, broadcast a route request (RREQ). However, the broadcast is sent to the multicast group IP address with the join flag "J" [8]. The nodes with most new sequence number respond to the RREQ with an RREP in the multicast group. The process to keep the multicast route tables, backward pointers are set [8].

After the expiration of the discovery period, the route is activated using a Multicast Activation (MACT) message unicast by the source to its particular next hop [8].

The routes must be kept until they are active. During the routes lifetime, due to the mobility of the nodes, broken links may occur and will be detected.

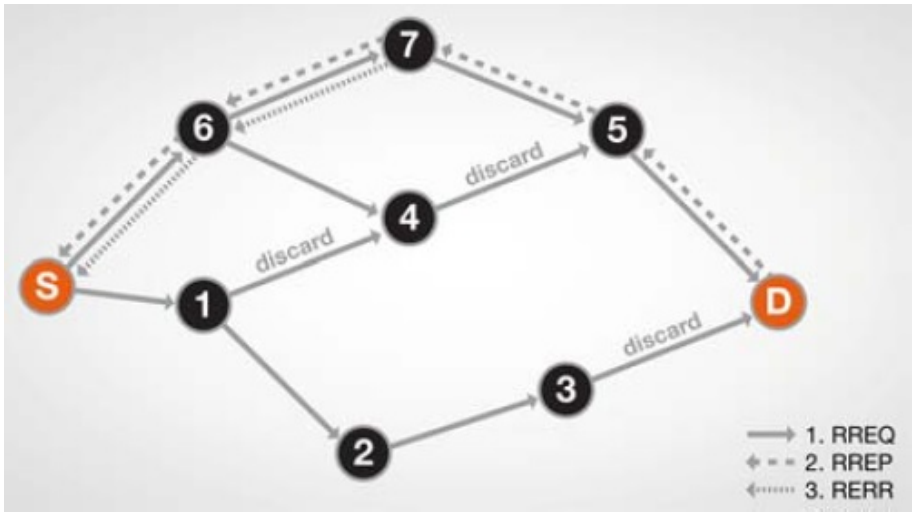


Figure 3.23: AODV routing example [8]

The working principle of AODV algorithm explained with an example is shown in Figure 3.23.

According to AODV algorithm, assume that the route is created from source (S) to destination (D) via nodes: 6, 7 and 5 called the forward route (RREQ). The destination (D) responds to the source (S) and creates the reverse route (RREP) [8]. However, if node 5 moves away from node 7, a link will be broken. Then, after detecting the broken link, node 7 broadcasts RERR back to the source (S).

Due to the mobility of node or bad conditions of the channel, if route re-discovery process is too regular, it effects on the performance [64].

4 System Architecture and Implementation

This chapter provides a system architecture and implementation of the proposed concept.

Environment

The OS Raspbian Strech is running on Raspberry Pis (Model : Raspberry Pi 3 Model B) and the processor 4 x ARM Cortex-A53 processor CPU 900MHz quad-core with 1 GB RAM. Each Raspberry Pi will be having a PLANetary library core implemented on them. For the communication between Raspberry Pis used Wi-Fi Direct and routing algorithm is used to know which Pi communicates with which other Pi.

PLANetary desktop application is running on PC to post queries in the sensor network and gets the result back, which is connected to the PLANetary or sink node over the Ethernet cable.

Procedure

1. Implementation of wireless communication technology between Raspberry Pis using Wi-Fi Direct
2. Implementation of PLANetary node or Sink node, where desktop application query and gets the result set.
3. Implementation of the Car2X scenario using the PLANetary library.

Figure 4.1 gives a detailed approach to the proposed concept.

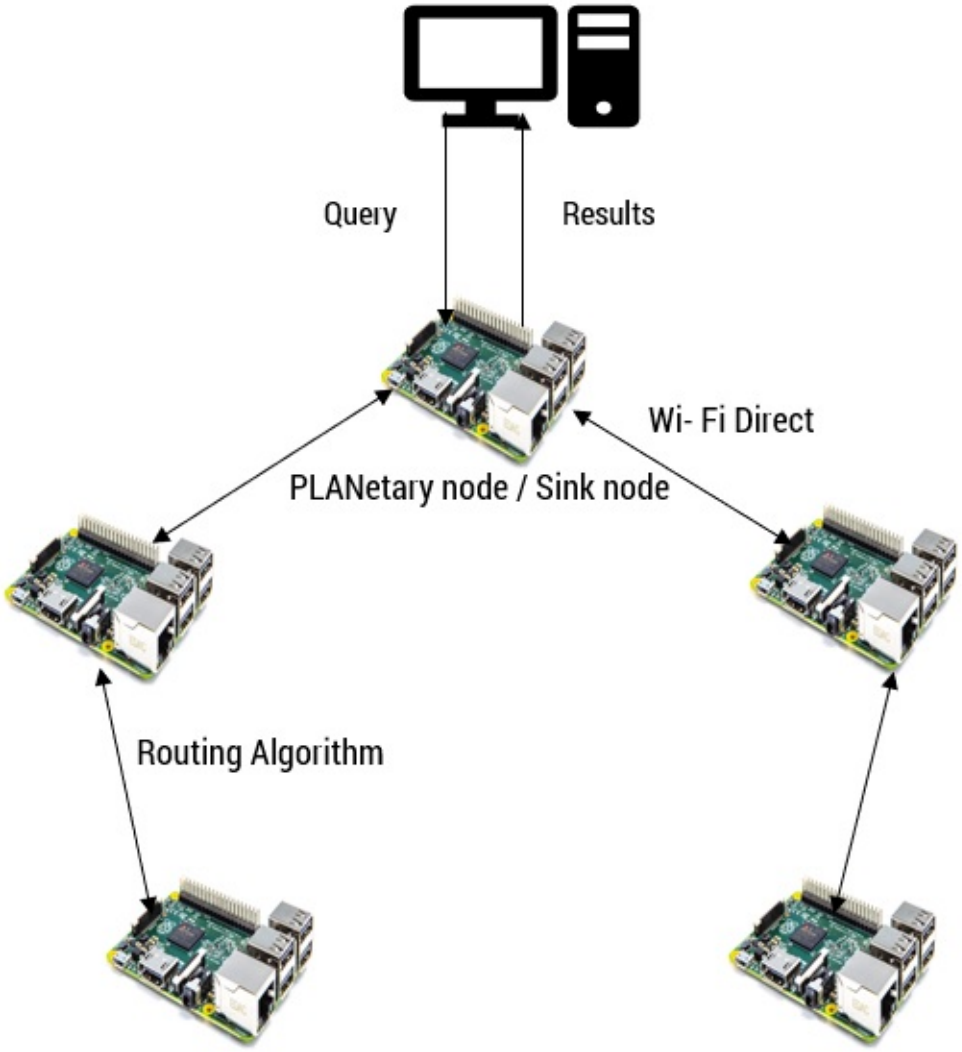


Figure 4.1: System Architecture

4.1 Wireless Communication between Raspberry PIs

4.1.1 Wi-Fi Protected Setup (WPS)

The Wi-Fi Protected Setup (WPS) standard is introduced to ease the deployment of secure Wi-Fi networks for the insertion of additional devices over time without any complications [65].

4.1.1.1 WPS methods

WPS, developed by the Wi-Fi Alliance, is a standard used for adding new Wi-Fi devices to the network. WPS methods are [65]:

1. PIN method
2. Push Button Control method (PBC method)

PIN method

The PIN has to be read from display on the new wireless device. This eight-digit PIN is generated by any network device. Usually, P2P GO generates the PIN and entered at P2P Client to establish a connection. The PIN method must be supported by the Wi-Fi Direct specification devices with a keypad or display must.

Push Button Control method (PBC method)

Instead of entering generated PIN by P2P GO, the user should push a button (actual or virtual) on both the P2P GO and the P2P Client [66]. After the connection is established, discovery mode turns off immediately or after some delay in most of the Wi-Fi Direct devices. Support of this mode is must for P2P GOand optional for connecting devices (P2P Clients).The PBC method must be supported by the Wi-Fi Direct specification devices with a keypad or display must.

P2P group is shown in Figure 4.2

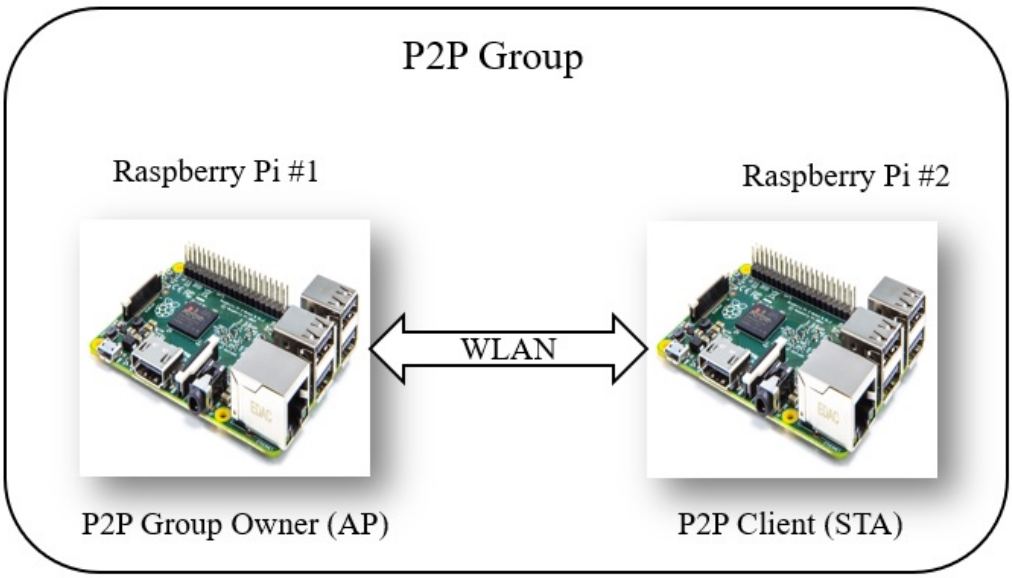


Figure 4.2: P2P Group diagram

4.1.2 P2P use cases

1. Connect in Pin (PIN Number) where Raspberry #1 is defined as the Group Owner (GO)
2. Connection using PIN code
3. Connect in PBC (Push Button Control)
4. Connect in PBC (Push button Control) where Raspberry #1 is defined as the Group Owner (GO)

1. Connect in Pin (PIN Number) where Raspberry #1 is defined as the Group Owner (GO)

Steps	Raspberry Pi #1	Raspberry Pi #2
1	run WPA Supplicant and WPA CLI	run WPA Supplicant and WPA CLI
2	`wpa_cli p2p_find`	`wpa_cli p2p_find`
3	`wpa_cli p2p_peers`	`wpa_cli p2p_peers`
4	`wpa_cli p2p_group_add`	
5	`wps_pin any`	
6		`p2p_connect Pi#1_MAC_ADDRESS Pi#1_PIN_NUMBER join`
7	Define IP Address for `p2p-wlan0-0`	Define IP Address for `p2p-wlan0-0`
8	Ping IP Address of Pi #2	Ping IP Address of Pi #1

Table 4.1: PIN method with GO

Raspberry Pi #1 establishes a Wi-Fi Direct AP (p2p-wlan0-0), Raspberry Pi #2 connects with it.

Raspberry Pi #1

Step 1 Running WPA supplicant and configuration is shown in Figure 4.3.

```
$sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
```

Channel = 1 and Go Intent value = 15. Because Raspberry Pi #1 defined as GO.

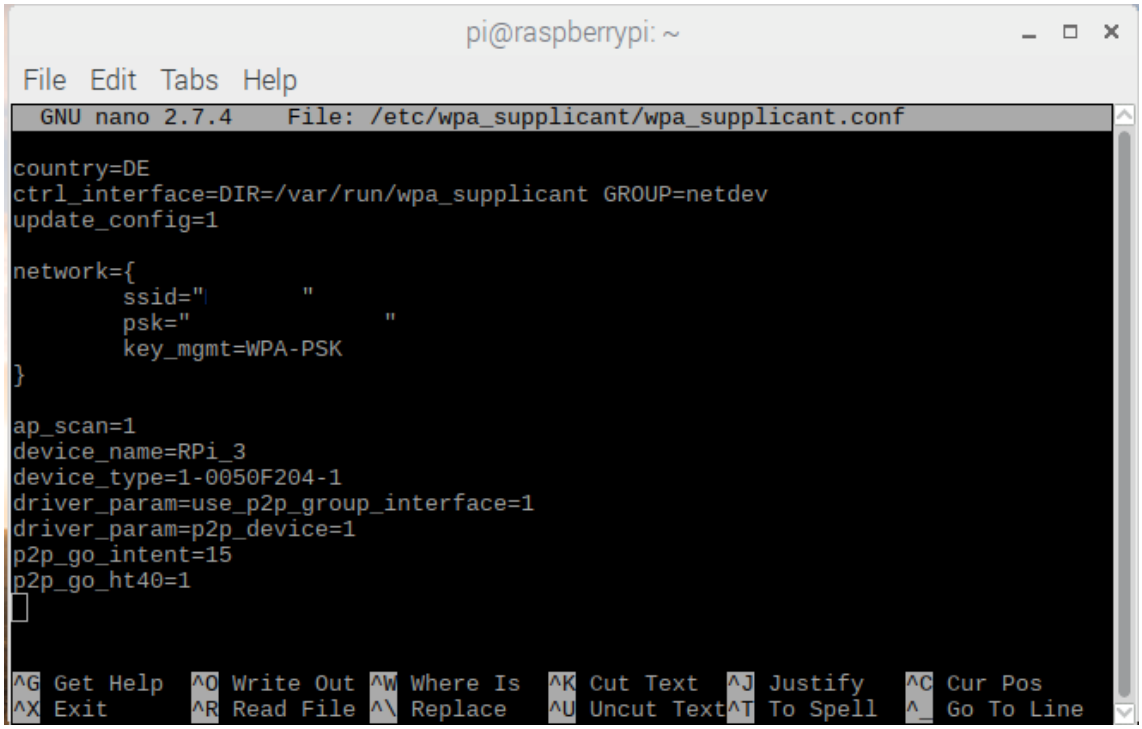


Figure 4.3: WPA Supplicant configuration on Raspberry Pi #1

Then run the command

```
$sudo wpa_supplicant -B -dd -iwlan0 -Dnl80211 -c
/etc/wpa_supplicant/wpa_supplicant.conf
```

Steps from 2 to 8 shown in Figure 4.4.

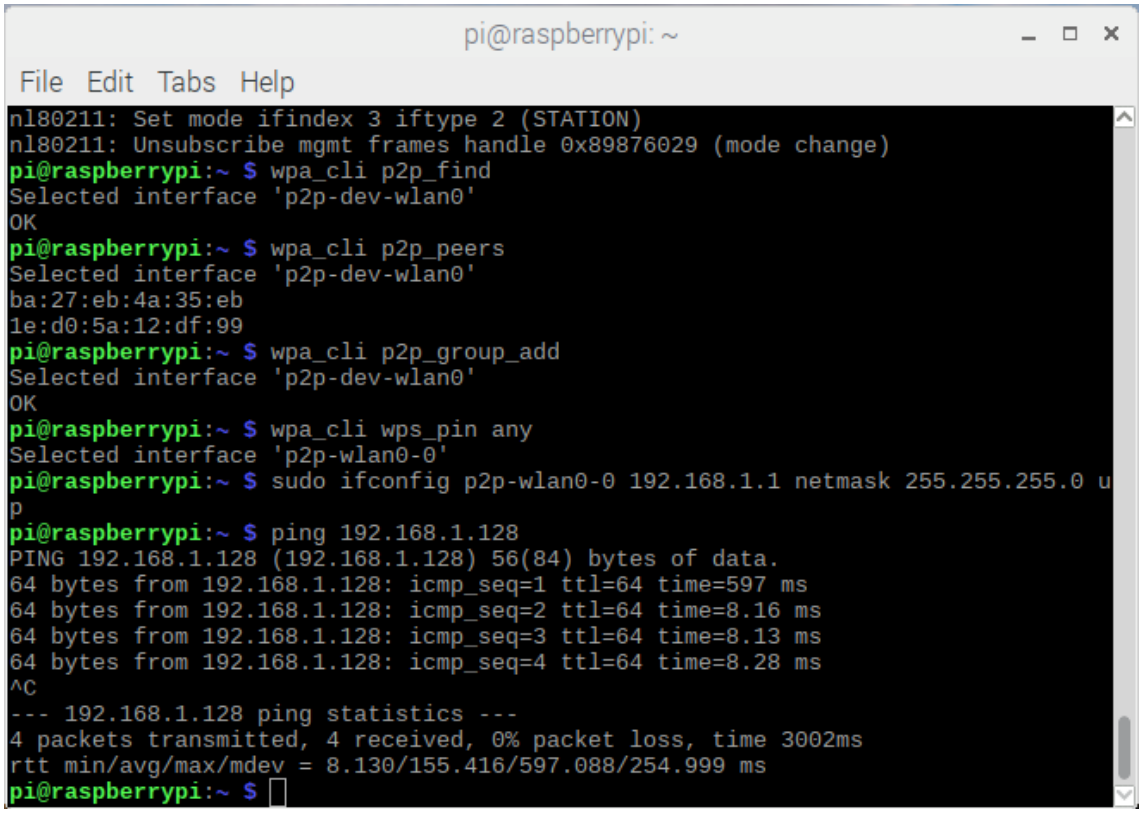


Figure 4.4: PIN method configuration on Raspberry Pi #1

Raspberry Pi #2

Step 1 Running WPA supplicant and configuration is shown in Figure 4.5.

```
$sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
```

Channel = 1 and Go Intent value = 0. Because Raspberry Pi #2 defined as CLIENT that connects to GO.

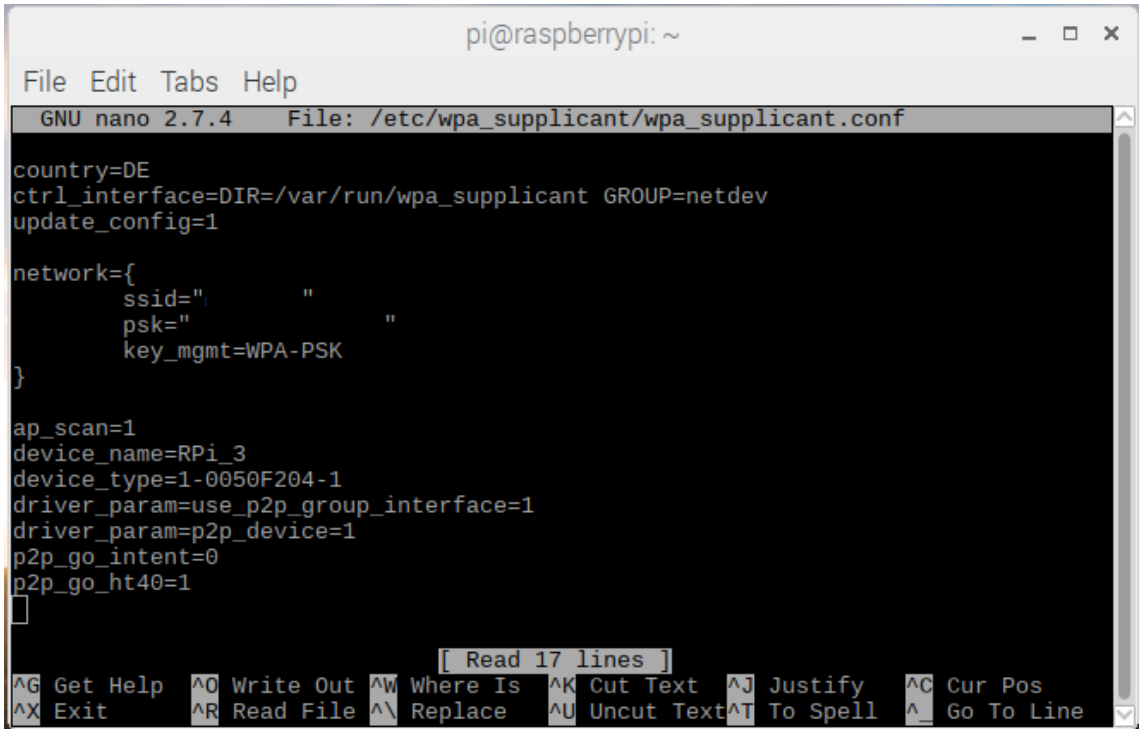


Figure 4.5: WPA Supplicant configuration on Raspberry Pi #2

Then run the command

```
$sudo wpa_supplicant -B -dd -iwlan0 -Dnl80211 -c
/etc/wpa_supplicant/wpa_supplicant.conf
```

Steps from 2 to 8 shown in Figure 4.6

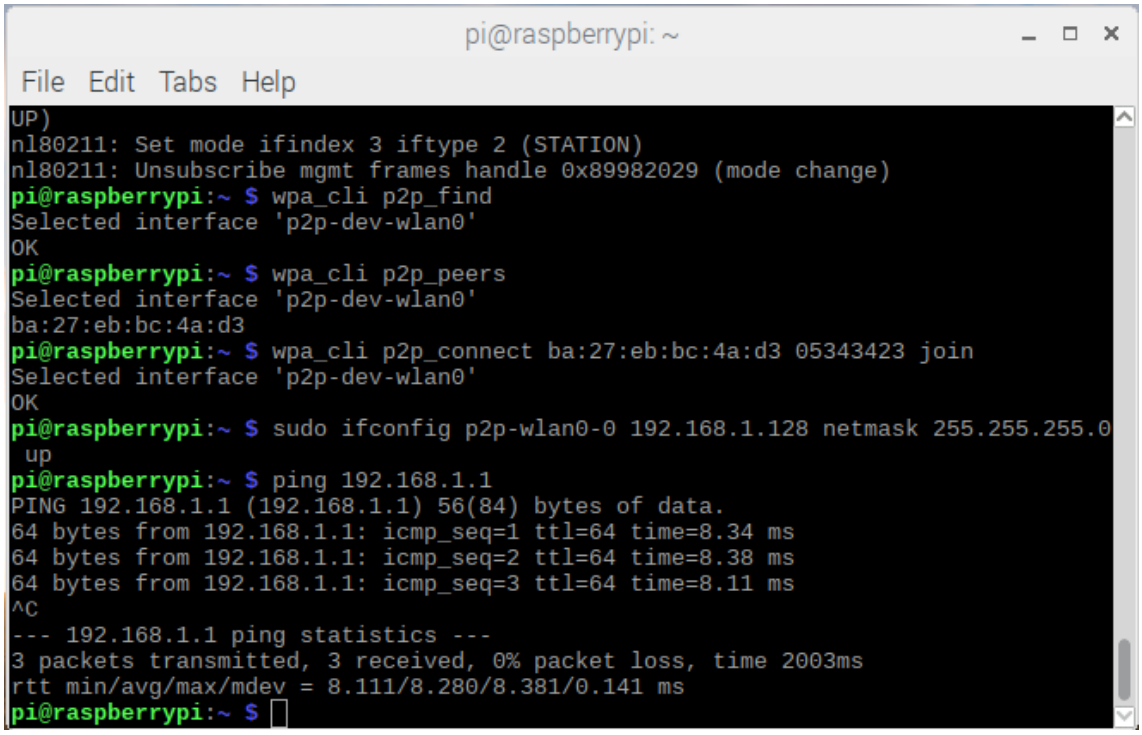


Figure 4.6: PIN method configuration on Raspberry Pi #2

Connection using PIN code

Steps	Raspberry Pi #1	Raspberry Pi #2
1	run WPA Supplicant and WPA CLI	run WPA Supplicant and WPA CLI
2	`wpa_cli p2p_find`	`wpa_cli p2p_find`
3	`wpa_cli p2p_peers`	`wpa_cli p2p_peers`
4	`wpa_cli p2p_connect Pi#2_MAC_ADDRESS pin auth`	
5		`wpa_cli p2p_connect Pi#1_MAC_ADDRESS Pi#1_PIN_NUMBER`
6	Define IP Address	Define IP Address
7	Ping IP Address of Pi #2	Ping IP Address of Pi #1

Table 4.2: Connection using PIN code

Connect in PBC (Push Button Control)

Steps	Raspberry Pi #1	Raspberry Pi #2
1	Set static IP Address	Set static IP Address
2	Run WPA Supplicant	Run WPA Supplicant
3	`wpa_cli`	`wpa_cli`
4	`p2p_find`	`p2p_find`
5	`p2p_peers`	`p2p_peers`
6	`p2p_connect Pi#2_MAC_ADDRESS pin auth go_intent=7`	
7		`p2p_connect Pi#1_MAC_ADDRESS pbc`
8	`exit wpa_cli`	`exit wpa_cli`
9	Ping IP Address of Pi #2	Ping IP Address of Pi #1

Table 4.3: Connect in PBC (Push Button Control)

Connect in PBC (Push button Control) where Raspberry #1 is defined as the Group Owner (GO)

Steps	Raspberry Pi #1	Raspberry Pi #2
1	Run WPA Supplicant	Run WPA Supplicant
2	`wpa_cli`	`wpa_cli`
3	`p2p_find`	`p2p_find`
4	`p2p_peers`	`p2p_peers`
5	`p2p_group_add`	
6	`wps_pbc`	
7		`p2p_connect Pi#1_MAC_ADDRESS pbc join`
8	`exit wpa_cli`	`exit wpa_cli`
9	Set static IP Address	Set static IP Address
10	Ping IP Address of Pi #2	Ping IP Address of Pi #1

Table 4.4: Connect in PBC (Push Button Control) with GO

After both Raspberry Pis connected with each other. Data transmission is possible with socket programming (e.g. by UDP).

Data transmission code for

P2P Server

```
  /* Create socket for sending/receiving datagrams */
  if ((sock = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0)
    printf("socket() failed");

  /* Construct local address structure */
  memset(&servAddr, '\0', sizeof(servAddr));
  servAddr.sin_family = AF_INET;
  servAddr.sin_addr.s_addr = htonl(INADDR_ANY);
  servAddr.sin_port = htons(ServPort);

  /* Bind to the local address */
  if (bind(sock, (struct sockaddr *) &servAddr, sizeof(servAddr)) < 0 )
    printf("bind() failed");

  for (;;)
  {
    /* Set the size of the in-out parameter */
    cliAddrLen = sizeof(clntAddr);

    /* Block until receive message from a client */
    if ((recvBufferSize = recvfrom(sock, Buffer, MAXSIZE, 0, (struct
   sockaddr *) &clntAddr, &cliAddrLen)) < 0)
      printf("recvfrom() failed") ;

    printf("Handling Client: %s\n", inet_ntoa(clntAddr.sin_addr));

    printf("[+]Data Received: %s",  Buffer);

    /* Send received datagram back to the client */
    if (sendto(sock, Buffer, recvBufferSize, 0, (struct sockaddr *) &
   clntAddr, sizeof(clntAddr)) != recvBufferSize)
      printf("sendto() Failed");
    printf("[+]Data Send: %s", Buffer);


  }
}
```

P2P Client

```
  /* Create a Datagram socket */
2 if ((sock = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0)
    printf( "socket () failed");
4
  /* Construct the server address structure */
6 memset(& servAddr, '\0', sizeof(servAddr));
  servAddr.sin_family = AF_INET;
8 servAddr.sin_addr.s_addr = inet_addr("192.168.1.1");
  servAddr.sin_port = htons(ServPort);
10
  /* Send the string to the server */
12 strcpy( Buffer, "Hello! P2P GO \n");
  if (sendto(sock, Buffer, MAXSIZE, 0, (struct sockaddr *) &servAddr,
   sizeof(servAddr)) < 0 )
14   printf("sendto() Failed");
  else
16   printf("[+]Data Send: %s", Buffer);

18 /* Receive a response */
  fromSize = sizeof(fromAddr);
20 if ((recvfrom(sock, Buffer, MAXSIZE, 0, (struct sockaddr *) &fromAddr
   , &fromSize)) < 0)
    printf("recvfrom() failed") ;
22
  if (servAddr.sin_addr.s_addr != fromAddr.sin_addr.s_addr)
24 {
    fprintf(stderr,"Error: received a packet from unknown source \n");
26  exit(1);
  }
28
  printf("[+]Data Received: %s\n", Buffer);
30 close(sock);
  exit(0);
32 }
```

Here P2P GO and P2P Client exchange the information.

4.2 Implementation of PLANetary node on Raspberry Pi

The implementation of the procedure consists of two parts.

1. PLANetary node (which includes PLANetary query system library) that runs on Raspberry Pi
2. Client software on a PC (i.e. PLANetary desktop application) through which the user feeds queries into the sensor network and can receive results from them.

However, PLANetary library and the desktop software to post queries to the sensor network are provided by Professorship for Computer Engineering at Technische Universität Chemnitz.

The main focus is to implement a PLANetary node on Raspberry Pi.

All the source files that implement PLANetary node are listed in Table 4.5 with their function. All architecture-dependent code needed for the query system is in the file `main.c`, which includes, for instance, code for reading the packets received from PC. The code for sending data packets (forwarding of queries, sending of the result set) is in the `main.c` file. In addition, the behavior and memory requirements of the system can be changed in the file `planetary_config.h`, that includes, for instance, the maximum number of simultaneously running queries `(MAX_RUNNING_QUERIES)` or whether the node should summarize result sets (for leaf nodes or nodes with very low resources, memory and computation time can be saved). The identification values for sensors and actuators of the nodes required for the evaluation of queries and for generating the results must also be defined by the user in the file `planetary_config.h`.

All other values for joining and comparison operators and aggregate functions are defined in the file `querytypes.h`.

The functions to handle and create planetary packets are defined in the file `packet_man.h` and `packet_man.c` and in the files comm.h and comm.c supporting functions are defined for the user.

Source File	Functionality
querytypes.h	Definitions of PLANetary
planetary_config.h	Parameters for changes and configuration of the query system
comm.h main.c	Functions implemented by the user
queries.h queries.c	Core of the system, such as reading and generating of data packets
conditions.h conditions.c	The logic for evaluating conditions and condition groups
grouping.h grouping.c	Functions for aggregating and grouping of the result data
utils.h utils.c	Supporting functions
packet_man.h packet_man.c	Functions to handle and create planetary packets

Table 4.5: Source code files for PLANetary functionality

4.2.1 Features of Server Application

The client program that controls the sensor network by the following features

- Connection to the data sink via a suitable interface (i.e. Ethernet cable)
- Simple definition of queries by the user
- Feeding queries into the sensor network and receiving results
- Overview of current and completed queries and their results

The connection of the Sink node to a PC running the client application is carried out via the interface Ethernet.

4.2.2 The Handshake between PLANetary node and PC

The PLANetary desktop application and Sink node on Raspberry Pi are connected over UDP as shown in Figure 4.7

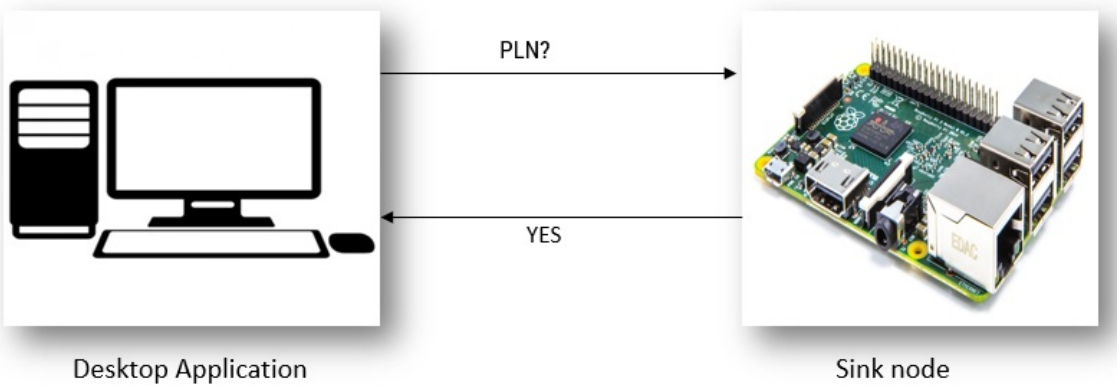


Figure 4.7: Connection between PLANetary Desktop application and Sink node

Raspberry Pi listens on UDP port 5000, where Desktop application listens on UDP port 32000.

Once start running the desktop application, the window is popped out for connection. Where UDP IP address, port and connect button will appear. IP address and Listening port of Raspberry Pi have to be entered. Then click on connect, it sends a packet 5000 port over UDP. The content of every packet is 4 maximum bytes (255,255,255,255), 1 byte of payload length (4) and payload `(PLN?)`.

When Raspberry Pi receives `PLN?` from a desktop application, then it should respond with the packet on port 32000 over UDP. The content of response packet is 4 maximum bytes, 1-byte payload length, and payload (i.e. `YES`).

```
#define SRC_PORT 5000
#define DST_PORT 32000
```

Desktop application before handshake is shown in Figure 4.8

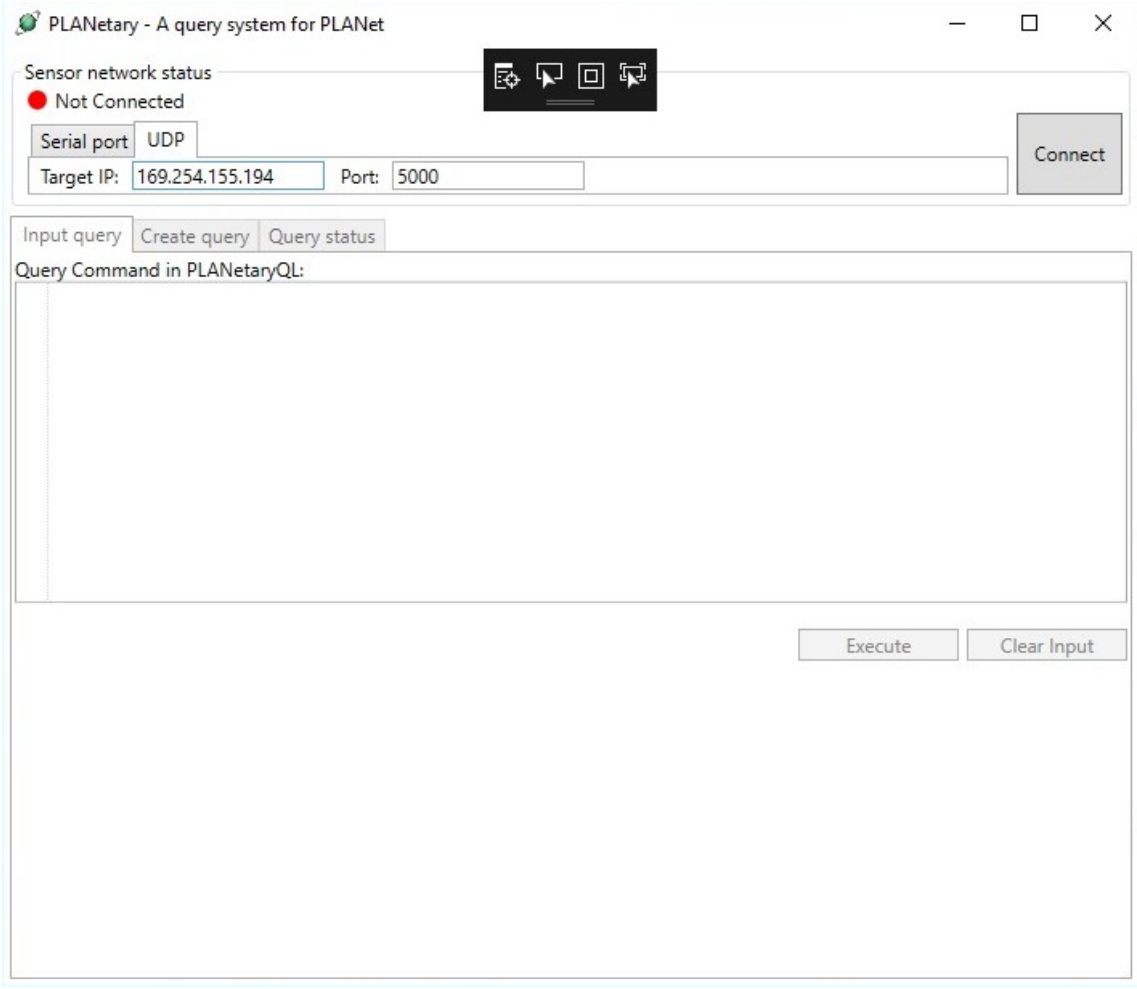


Figure 4.8: Desktop application before handshake

Handshake code

```
printf("Launching PLANetary sink node \n");


if ((sock = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0)
  printf("socket() failed");


memset(&servAddr, 0, sizeof(servAddr));
servAddr.sin_family = AF_INET;
servAddr.sin_addr.s_addr = htonl(INADDR_ANY);
servAddr.sin_port = htons(SRC_PORT);

memset(&clntAddr, 0, sizeof(clntAddr));
```

```
14    clntAddr.sin_family = AF_INET;
      clntAddr.sin_addr.s_addr = inet_addr("169.254.93.95");
16    clntAddr.sin_port = htons(DST_PORT);


18
      if (bind(sock, (struct sockaddr *) &servAddr, sizeof(servAddr)) < 0)
20      printf("bind() failed");

22    for (;;)
      {
24      printf("Listening on UDP:5000 \n");


26
        cliAddrLen = sizeof(clntAddr);
28


30      if ((recvMsgSize = recvfrom(sock,  recBuffer, MAXSIZE, 0,(struct
       sockaddr *) &clntAddr, &cliAddrLen)) < 0)
          printf("recvfrom() failed") ;
32
        printf("Handling Client: %s\n", inet_ntoa(clntAddr.sin_addr));
34
        printf("Received:");
36      for(i = 0; recBuffer[i] != '\0'; ++i)
          printf("%c",recBuffer[i]);
38      printf("\n");


40
        unsigned char respBuffer[] = {255, 255, 255, 255, 4, 'Y', 'E', 'S'
       };
42      if (sendto(sock,  respBuffer, sizeof(respBuffer), 0, (struct
       sockaddr *) & clntAddr, sizeof(clntAddr)) < 0)
          printf("sendto() failed");
44

      }
46

}
```

Received packet on Raspberry Pi over UDP is shown in Figure 4.9

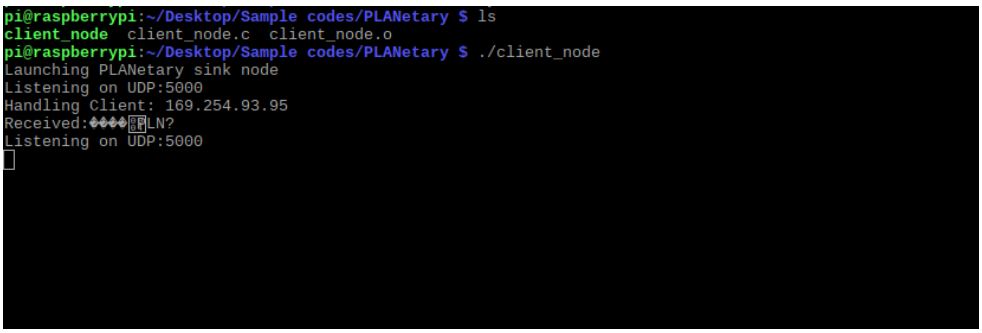


Figure 4.9: Received packet PLN? from a desktop application

Desktop application after handshake is shown in Figure 4.10

Connected to Sink node on Raspberry Pi over UDP

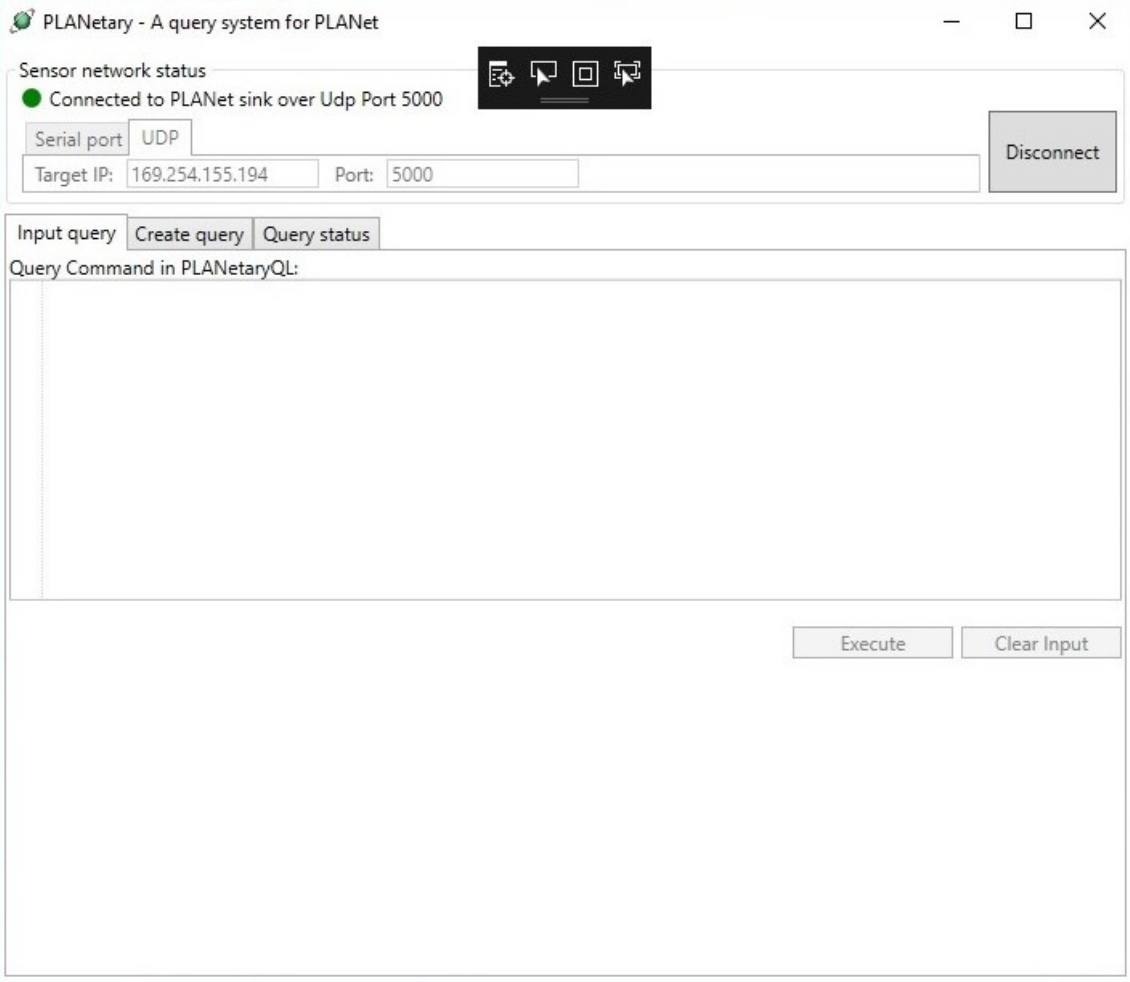


Figure 4.10: Desktop application after handshake

Receiving and Sending Packets

The sequence diagram for receiving and sending packets is shown in Figure 4.11.

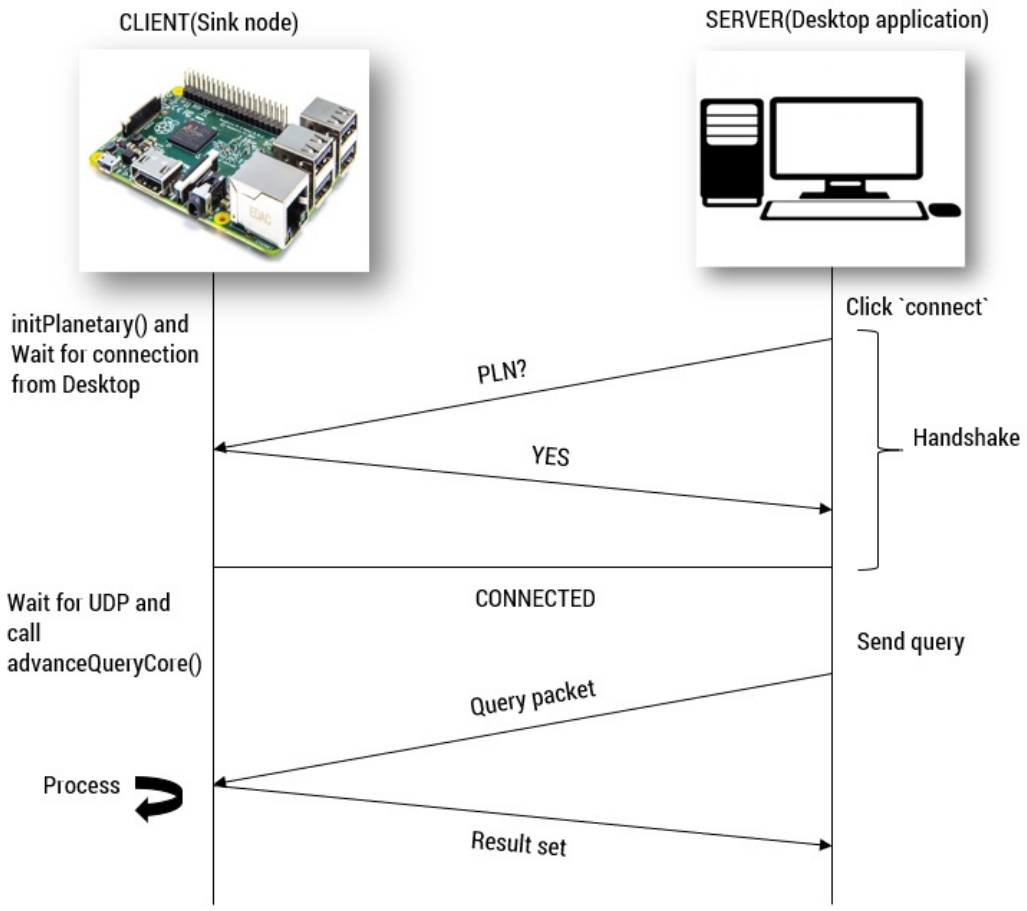


Figure 4.11: Sequence diagram for receiving and sending packets

In this case, Desktop application is a server and Sink node is a client. On Raspberry Pi, having both PLANetary library (.c and .h files) and UDP handshake code.

Steps for Receiving and Sending packets

1. initPlanetary()
2. Wait for a connection from Desktop application - Handshake
3. Listen on UDP
4. Call `advanceQueryCore()`
5. Packet - receive on UDP to PLANetary
6. PLANetary send packet over UDP

PLANetary packet format

To forward queries to the sensor network, a data format is needed that can save all the information of the query in a memory efficient way. The data format is used to distribute queries through the wireless connection of the nodes within the propagation phase and is shown in Figure 4.12. Before it is executed, each query is given a unique ID in the sensor network so that when nodes send their result sets, nodes can reference the corresponding query to which the results belong.

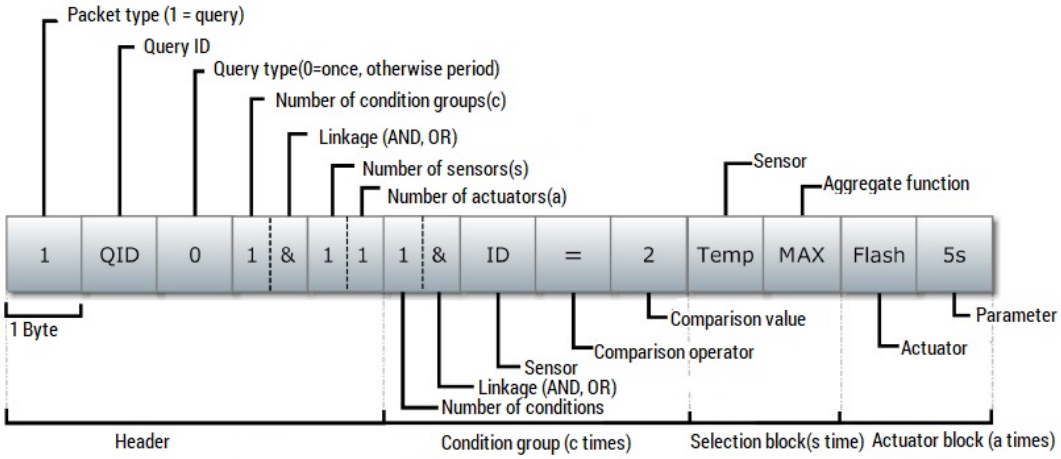


Figure 4.12: Structure of a packet that describes a query [20]

Sending query packet is shown in Figure 4.13

For Instance, `SENSE ID AT sensors`

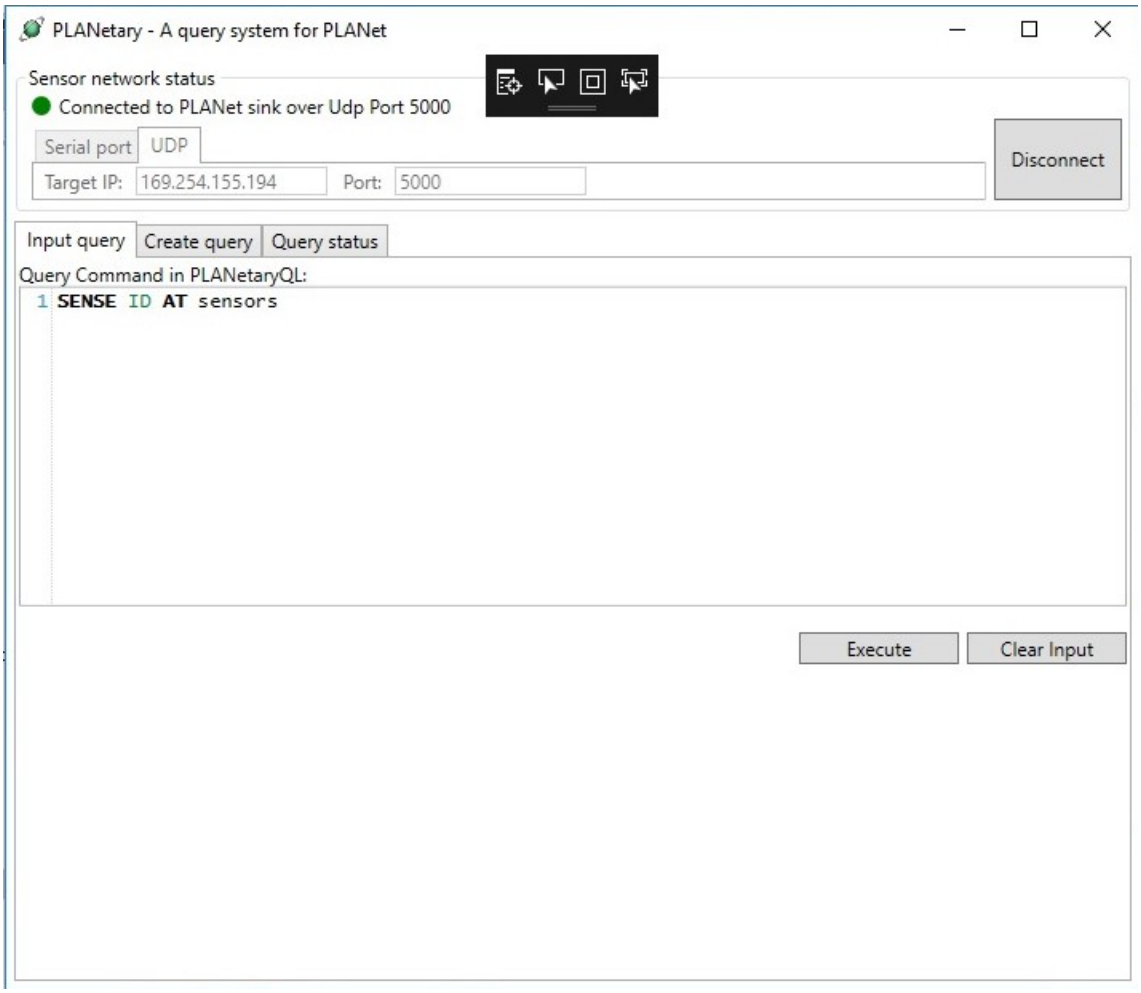


Figure 4.13: Sending query packet from a desktop application

Code for processing of incoming packets

```
void processIncomingPackets(QueryCore* core)
{
    // Handles incoming packets
    struct sockaddr_in fromAddr;
    unsigned char packetBuff[RX_BUF_SIZE];
    int recvPacketSize;

    // NON_BLOCKING Call
    fd_set sockSet;
    fcntl(sock, F_SETFL, O_NONBLOCK);
    struct timeval selTimeout;
```

```
12      int result;

14      FD_ZERO(&sockSet);
        FD_SET(sock, &sockSet);
16
        selTimeout.tv_sec = 20;
18      selTimeout.tv_usec = 0;

20      result = select(sock + 1, &sockSet, NULL, NULL, &selTimeout);

22      if (result < 0)
        FD_ISSET(sock,&sockSet);
24
    if (result == 1)
26      recvPacketSize = recvPlanetPacket(packetBuff, RX_BUF_SIZE, &
      fromAddr);
    else
28       printf("TIMEOUT: recvfrom() failed \n") ;

30      if (recvPacketSize)
        {
32        LOG("Handle PLANetary Packet");
            handlePlanetaryPacket(core, &packetBuff[5], recvPacketSize);
34      }
}
```

Code for processing of outgoing packets

```
void processOutgoingPackets(QueryCore* core)
2 {

4       if (!queueIsEmpty(&core->packetQueue))
        {
6           LOG("Send some packets");
            while (!queueIsEmpty(&core->packetQueue))
8           {
                PacketToSend* sendInfo = queuePeekHead(&core->packetQueue);
10
                unsigned char responseBuf[TX_BUF_SIZE];
12              int responseLen = createPlanetaryPacket(core, sendInfo,
    responseBuf, TX_BUF_SIZE);
                if (responseLen > 0)
14              {
                    struct sockaddr* targetAddr;
16                  if (getNodeAddress(sendInfo->receivers[sendInfo->curPos
    ], &targetAddr))
                    {
18                      if (!sendPlanetPacket(targetAddr, responseBuf,
    responseLen))
                        LOG("Failed to send packet");
20                  } else
```

```
                    LOGV("Could not resolve node address of node id %d"
    , sendInfo->receivers[0]);
22              } else
                {
24                  LOG("Packet creation failed");
                }
26
                queueNext(&core->packetQueue);
28          }
        }
30  }
```

When the packet is received from the desktop application, it is given to the function `handlePlanetaryPacket()`. The call `recvfrom()` is non-blocking, which either receives a packet or returns an error. The `recvfrom()` is required to unblock after some amount of time to allow the client to handle the packet loss and also which allows to receive packet indefinitely. The function `handlePlanetaryPacket()` checks the received packet type, such as `PT_QUERY, PT_QUERY_RESULT` etc. Furthermore, it will read the packet and put the query in the internal memory.

Code for Handling planetary packet

```
1   void handlePlanetaryPacket(QueryCore* core, unsigned char* packetData,
        int packetLen)
    {
3       switch(packetData[0])
        {
5           case PT_QUERY:
                scheduleQuery(core, packetData);
7               break;
            case PT_QUERY_RESULT:
9               {
                    // query id
11                  int query_id = packetData[1];
                    Query* query;
13
                    if (findQuery(core, query_id, &query))
15                      readResultPacket(core, packetData, query);
                }
17          //default:
                //LOGV("Received unknown packet type: %d", packetData[0]);
19      }
    }
```

The function `advanceQueryCore()`, where the internal time which is moved forward inside of the PLANetary library is shown in Figure. It evaluates queries and checks if any queries have finished aggregating their results. Then `advanceQueryCore()` signals and returns true if there are finished queries.

Code for function advanceQueryCore()

```
int main()
{
    LOG("Initializing QueryCore");
    QueryCore qCore;
    init(&qCore);

    LOG("Launching PLANetary sink node");

    initSocket();

    // Address of the control application
    struct sockaddr_in parentAddr;

    // Handshake and SINK NODE Connection
    if (qCore.mode == AM_NODE || handshake(&parentAddr))
    {
        if (qCore.mode == AM_SINK)
        {
            LOG("Connected to Control Application");
            qCore.routingTable.parentId = 0;
            setNodeAddress(qCore.routingTable.parentId, (struct
    sockaddr*)&parentAddr);
        }

        // Main loop
        clock_t start = clock();
        while(true)
        {
            processIncomingPackets(&qCore);

            // Pass elapsed millis to advanceQueryCore
            clock_t end = clock();
            int millis_elapsed = (end - start) / (CLOCKS_PER_SEC*1000);

            bool queriesChanged = advanceQueryCore(&qCore,
    millis_elapsed);
            start = clock();

            if (queriesChanged)
                processOutgoingPackets(&qCore);
        }
    }



    return 0;
}
```

Then call `createResultPacket()`, it fills a response buffer. Then response buffer will be sent over UDP to the desktop.

Code for creating result packet

```
int createPlanetaryPacket(QueryCore* core, PacketToSend* packetInfo,
    unsigned char* buf, int bufLen)
{
    // send query results to parent
    if (packetInfo->sendResults)
    {
        return createResultPacket(core, packetInfo->query, buf);
    }

    // propagate cancellation of query
    if (packetInfo->cancelQuery) {
        buf[0] = packetInfo->query->id;
        buf[1] = PT_CANCEL_QUERY;
        return 2;
    }

    // propagate query to children
    return createQueryPacket(core, packetInfo->query, buf);
}
```

When a sensor node sends the result set of a query to its parent node, it uses the data format as shown in Figure 4.14. The results of the original query that the nodes hold because the number of result values per result row is not specified.

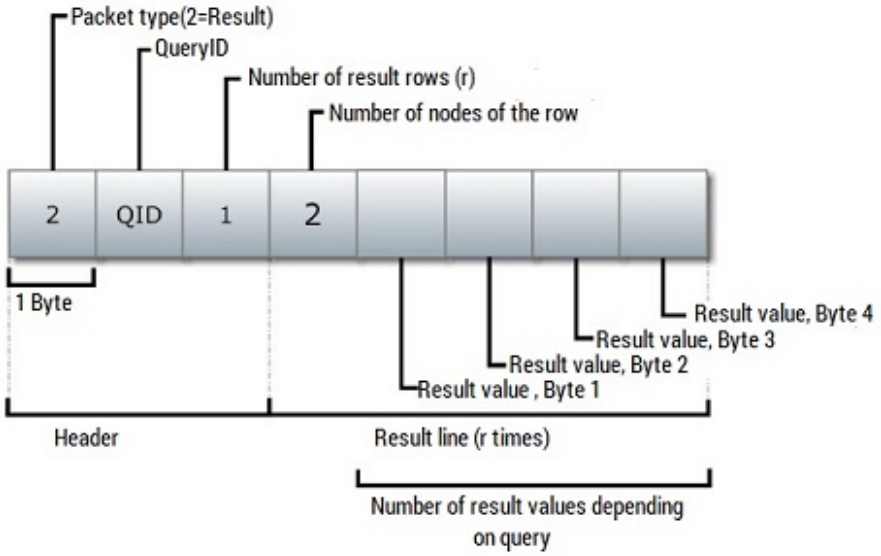


Figure 4.14: Structure of a packet that describes a result set [20]

The Figure 4.15 shows sending of result set from Raspberry Pi.

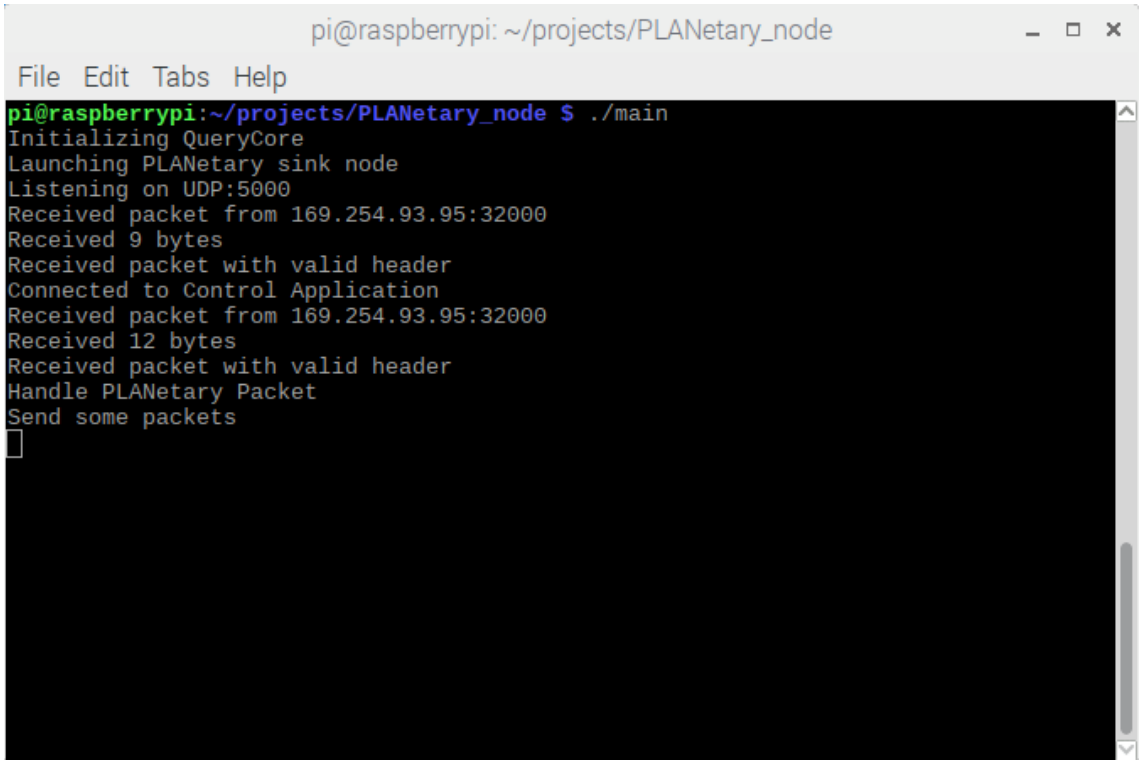


Figure 4.15: Sending result set from Raspberry Pi

The desktop is able to read this packet and it will show the result as shown in Figure 4.16

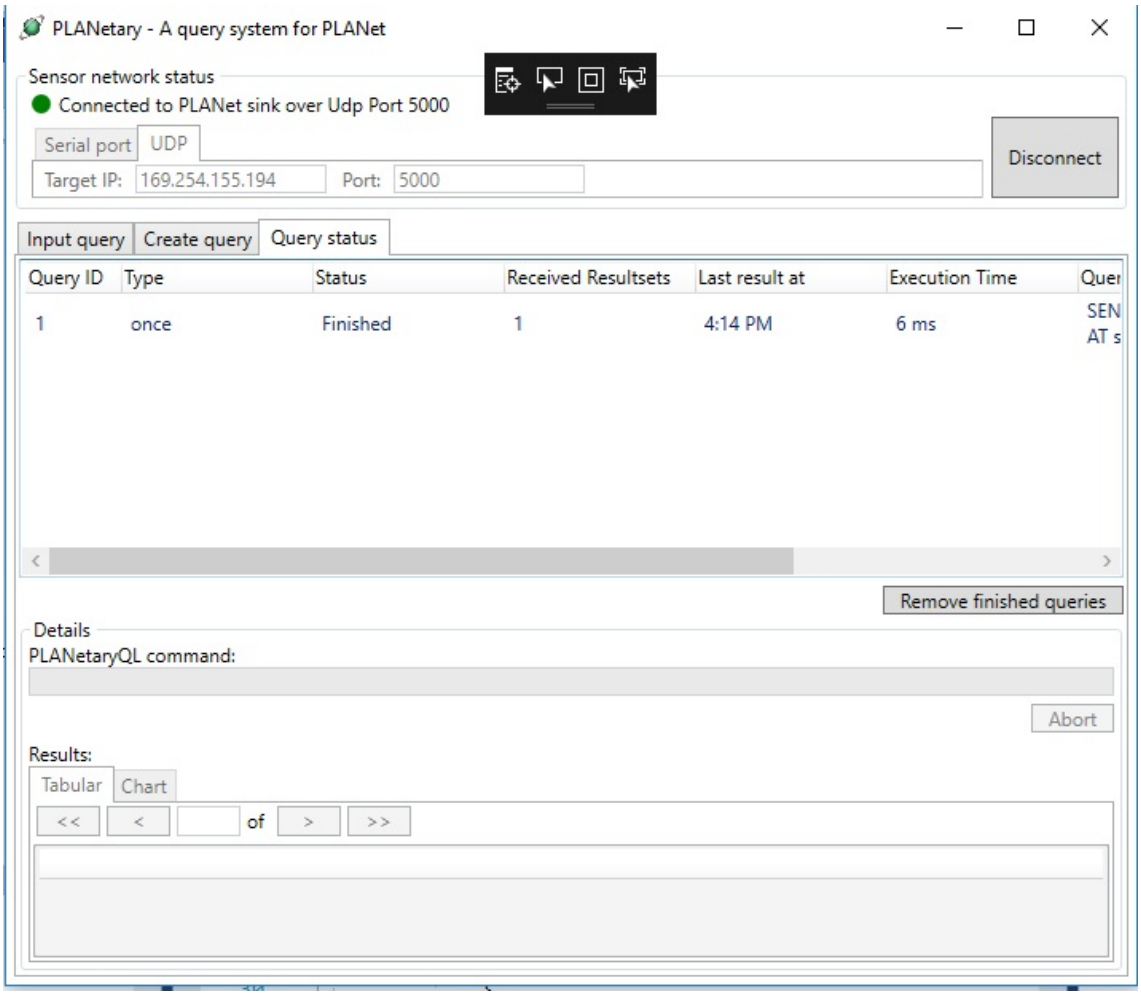


Figure 4.16: Result set on Desktop application

4.3 Implementation of Car2X scenario using PLANetary library

4.3.1 Car2X communication scenario use case

To implement Traffic congestion detection, Sensors that required are: speed of the vehicle and traffic density [24]. set of speed and traffic density estimates, level of congestion as shown in Table 4.6.

Level of congestion	Vehicle's speed	Traffic density
Slight	[48-81] km/h	[29-37] veh/km/ln
Moderate	[24-64] km/h	[37-50] veh/km/ln
Severe	Below 40 km/h	Above 50 veh/km/ln

Table 4.6: Level of congestion [24]

The traffic density and speed of the vehicle as input and provides the traffic congestion level as an output.

The vehicle's speed: can get from a GPS device or the CAN bus of the vehicle (km/h).

The traffic density: can get through the messages from neighboring vehicles (veh/km/ln). The traffic density depends on the number of neighboring vehicles detected, their distance to the vehicle measuring the traffic density, and the number of road lanes [24].

Moreover, all the vehicle can evaluate its local traffic conditions or congestion using the speed of the vehicle and traffic density, and then stores these values into the ITS [24].

4.3.2 Implementation of Car2X scenario

The sensor types are defined in `planetary_config.h` and `main.c`

Code for sensor types

```
// SENSOR TYPES
#define SENSOR_ID 0
#define SENSOR_VEHICLESPEED 1
#define SENSOR_TRAFFICDENSITY 2
```

```
int getSensorValue(int sensorId)
{
    switch(sensorId)
```

```
4       {
            case SENSOR_ID:
6               return 0;
            case SENSOR_VEHICLESPEED:
8               return 75;
            case SENSOR_TRAFFICDENSITY:
10              return 35;
            default:
12              return 0;
        }
14  }
```

A simple query, For instance

`SENSE Density AT sensors WHERE Density < 30` as shown in Figure 4.17. The required data would be collected and extracted at the sink node.

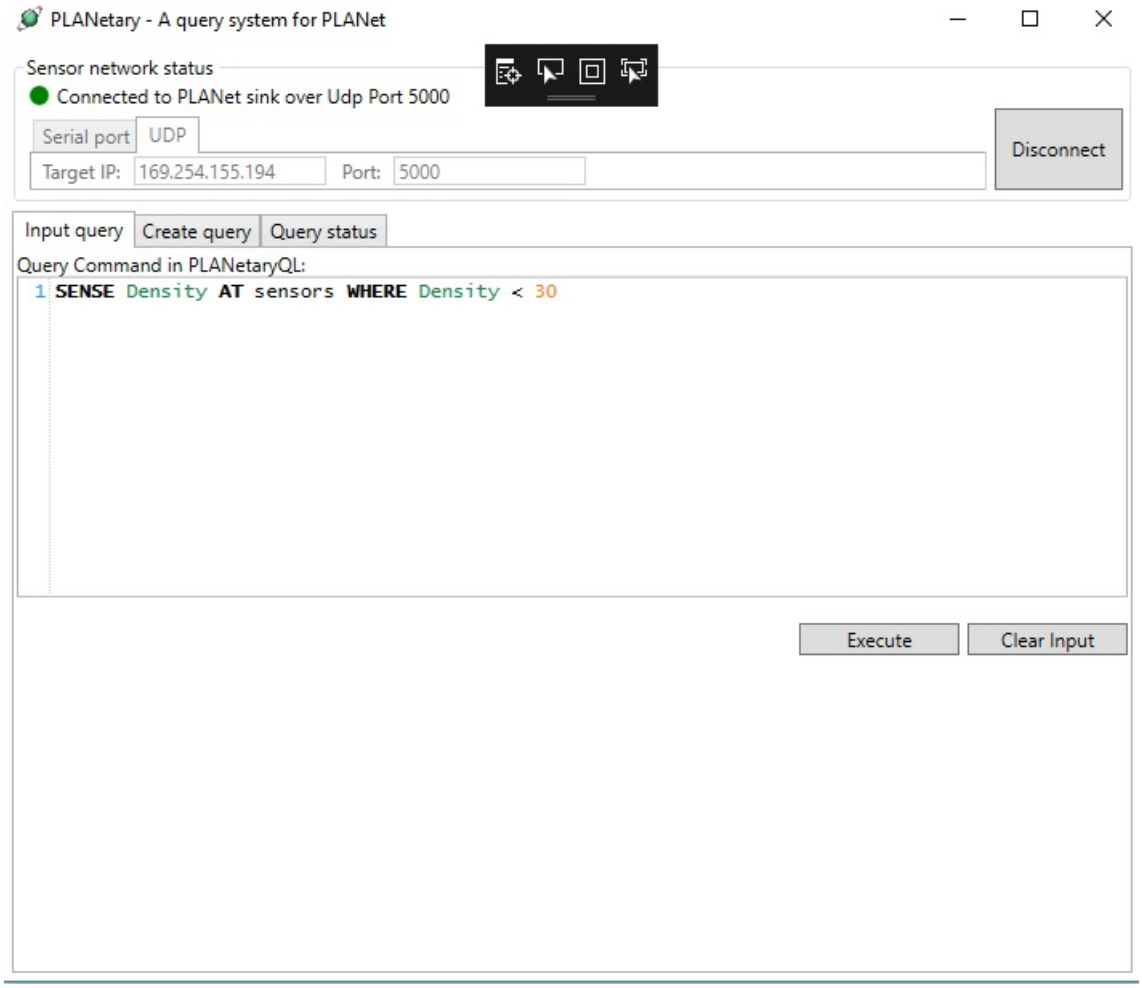


Figure 4.17: Querying network for Traffic Density

The result sets and query execution time can be seen in Figure 4.18.

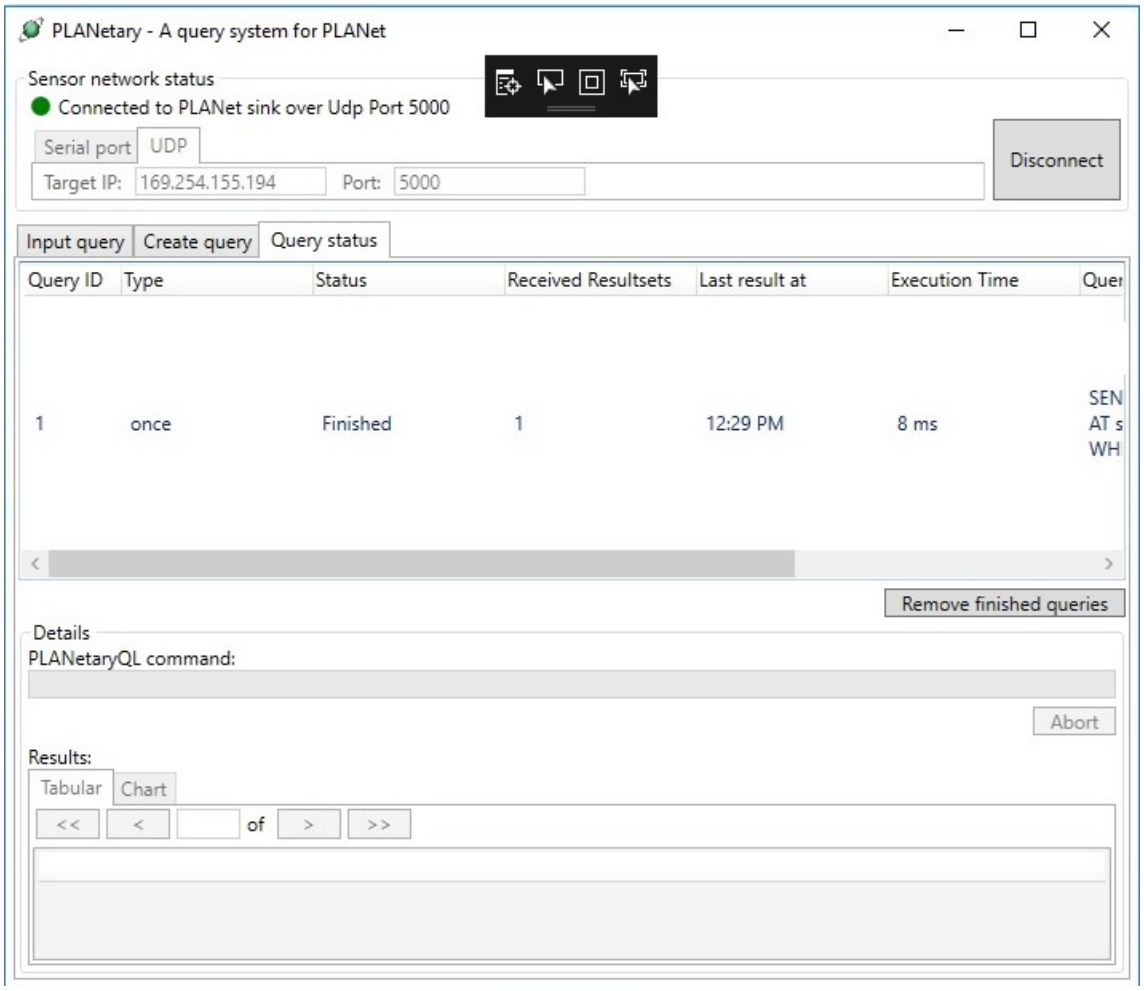


Figure 4.18: Response time for Traffic Density

5 Performance Evaluation

The approach is tested with a sensor network of five nodes (where one node is Sink node or PLANetary node) of the PLANet type which is a platform developed at the Professorship for Computer Engineering at Technische Universität Chemnitz [45].

5.1 Evaluation of Wi-Fi Direct

The approach is to implement communication between nodes without using an AP. Because of that using Wi-Fi Direct, where the nodes communicate with each other as explained in Chapter 2. Wi-Fi Direct device can operate at distances up to 200 m and data transfer speed is up to 250 Mbit/s. Compared with other wireless communication standards, configuration of the Pis is easy and low connection time.

In this thesis, Wi-Fi Direct could be used in Car2X communication scenarios. However, disadvantages of implementing Wi-Fi Direct are: a delay occurs in the discovery phase where the two P2P devices to find each other and delay occurs in the formation phase as well where the roles have been negotiated by the devices for GO and Client. If the GO leaves or lost connection, the group is collapsed and connection is lost between all clients of the group. The entire process has to start from beginning of assigning channels and intent values.

Wi-Fi Direct has limitations, to maintain a low delay, the group size should be limited and if the user needs to press a button on each node to initiate the connection this is not suitable.

Initially while using Raspberry Pi with NOOBSv2.4.4, the connections were working seamlessly. However, post the upgrade of Raspberry Pi distribution, although the interface `p2p-wlan0-0` was being created. There was a setback while connecting with other nodes. Where in the error encountered was `p2p-wlan0-0 departed`.

5.2 Evaluation of PLANetary

The approach is to measure query response time with each node using PLANetary libraries, by adding nodes one by one at a time.

A simple query, For example

```
SENSE Density AT sensors WHERE Density < 30
```

For this query, evaluating the latency for five nodes by adding one by one. Latency has been evaluated for the sink node as shown in Figure 4.18. However, for the remaining nodes have not been implemented and expected results would be as shown in Figure 5.1 .

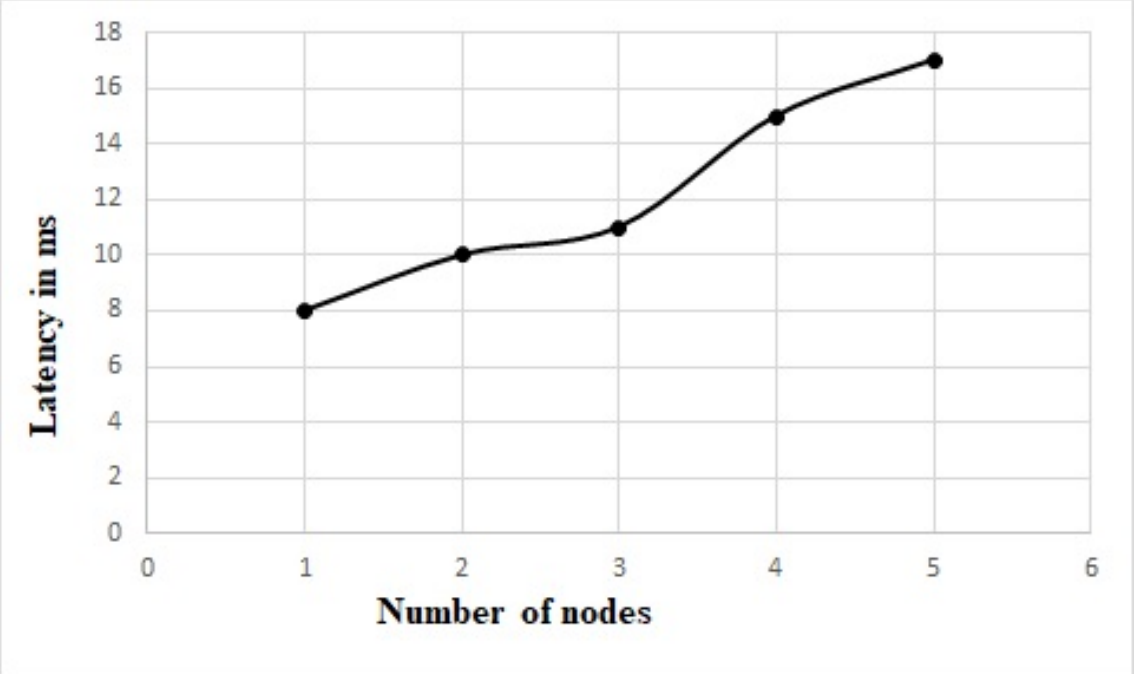


Figure 5.1: Query response time of nodes for a single query

Therefore, usually the latency of the query increases with increasing number of nodes.

Figure 5.2 shows the evaluation of latency on sink node with different queries. For the first query, latency is more. However, with increasing number of queries latency comes down.

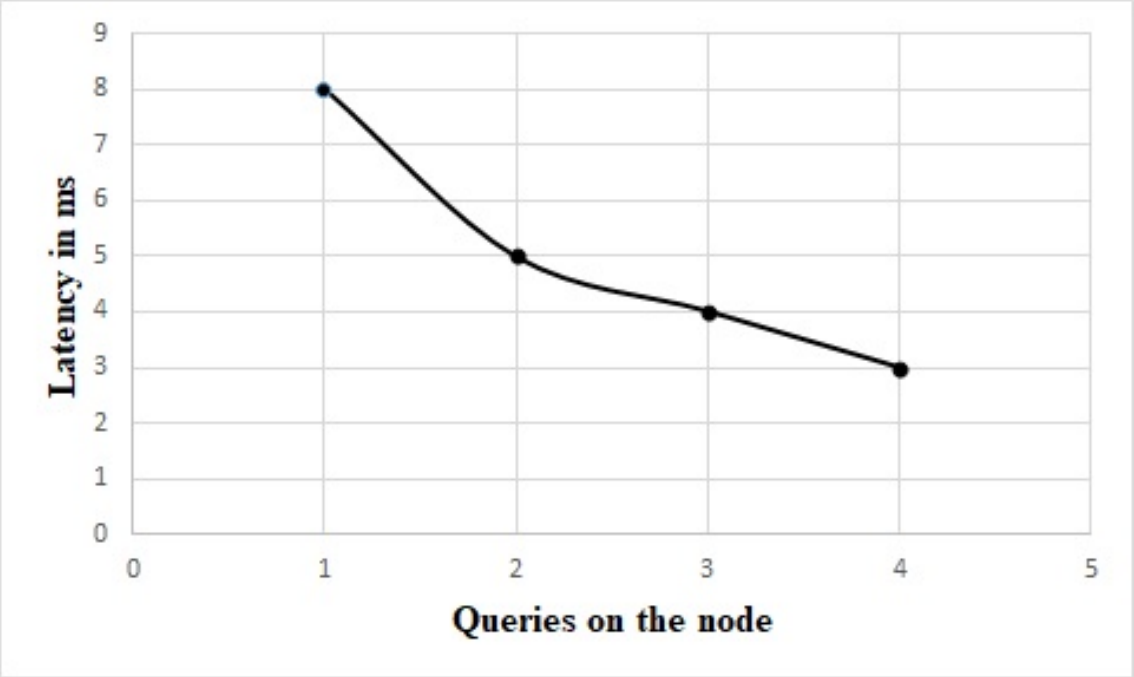


Figure 5.2: Query response time on sink node for distinctive queries

The query packet size (i.e. traffic volume in the network) for the queries on the sink node measured and shown in Figure 5.3.

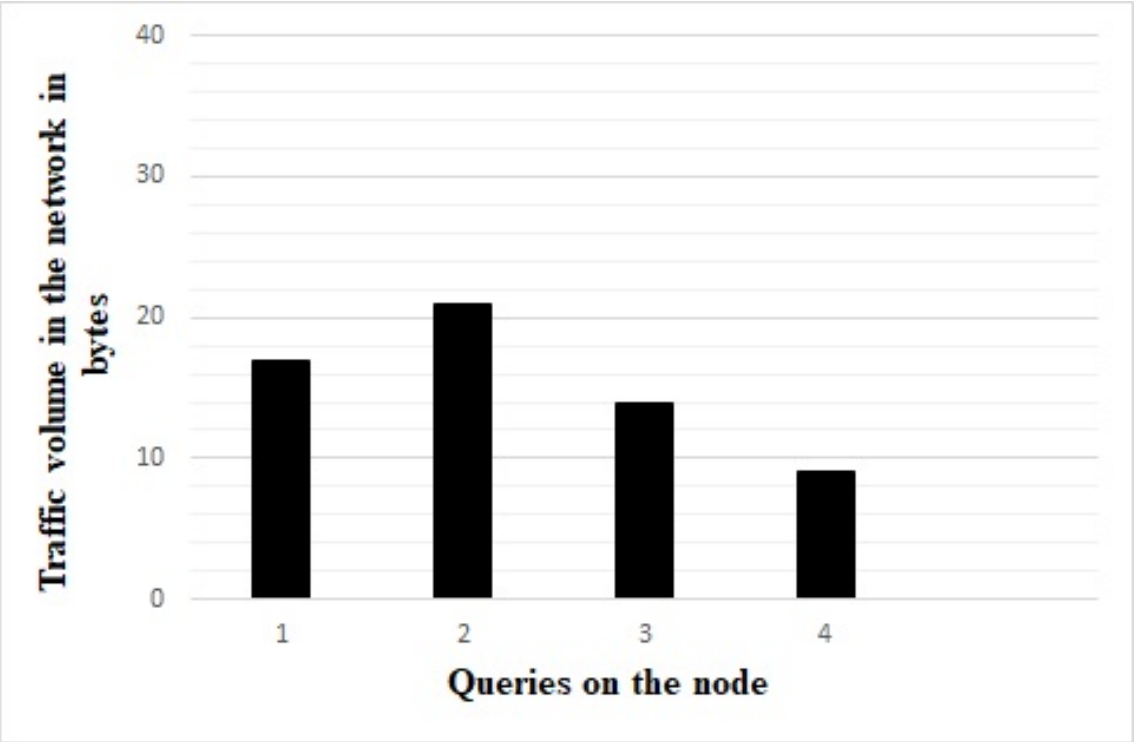


Figure 5.3: Traffic volume in the network for the queries

Generally, traffic volume in the network depends on the query packet size and result sets. The more different a query is, the more limited is the node subset on which it is executed and the fewer values need to be collected. Therefore, traffic volume decreases with increase in different queries.

6 Conclusion

This thesis described the concept and implementation of a Car2X communication application using a queriable wireless sensor network and presented an in-depth analysis of Wi-Fi Direct or Wi-Fi P2P wireless technology and implemented on Raspberry Pis. PLANetary query system has been introduced and implemented PLANetary node on Raspberry Pi. Lastly, it was evaluated whether the PLANetary library can be used for Car2X communication scenarios.

The Wi-Fi Direct was implemented on Raspberry Pis for the communication purpose using the specification from the Wi-Fi Alliances. Even though Wi-Fi Direct from the same family of Wi-Fi, it is different from tradition Wi-Fi networks like Infrastructure and Ad-hoc mode.

The database-oriented strategies are very useful in increasing the energy efficiency of embedded and WSNs. The PLANetary node application on a Raspberry Pi and the handshake between PC and Raspberry Pi communicated by Ethernet cable has been successfully implemented, where PC represents desktop application and Raspberry Pi represents the PLANetary node. PLANetary focuses on energy efficiency, light-weight architecture, and platform independent.

Furthermore, the thesis presents Car2X communication scenario with required sensors and their values and implemented using the PLANetary library. For instance, traffic congestion detection. In this scenario, using database-oriented approach, fuse queries in the network and gets the results accurately.

In the future, the sensor networks with thousands of nodes shall be evaluated and the deployment of a network with a higher number of nodes than the nodes used in this thesis.

The PLANetary library shall be used in other Car2X communication scenarios as well.

Appendix

Content in CD

Master Thesis Documents

`Master_Thesis_Report_Srinivasu_Jitta.pdf`
`Master_Thesis_Task_Description.pdf`
`Master_Thesis_Poster.pdf`

Implementation

PLANetary on Raspberry PI
`main.c`
`comm.h`
`comm.c`
`packet_man.h`
`packet_man.c`

Car2X scenario
`main.c`

Bibliography

[1] H. Stübing, *Multilayered security and privacy protection in Car-to-X networks: solutions from application down to physical layer*. Springer Science & Business Media, 2013.

[2] S. Yinbiao, K. Lee, P. Lanctot, F. Jianbin, H. Hao, B. Chow, and J. Desbenoit, "Internet of things: wireless sensor networks," *White Paper, International Electrotechnical Commission, http://www. iec. ch*, 2014.

[3] Y. He and A. Tully, "Query processing for mobile wireless sensor networks: State-of-the-art and research challenges," in *Wireless Pervasive Computing, 2008. ISWPC 2008. 3rd International Symposium on*. IEEE, 2008, pp. 518–523.

[4] J. Gehrke and S. Madden, "Query processing in sensor networks," *IEEE Pervasive computing*, vol. 3, no. 1, pp. 46–55, 2004.

[5] X. Jia, J. Wu, and Y. He, *Mobile Ad-hoc and Sensor Networks: First International Conference, MSN 2005, Wuhan, China, December 13-15, 2005, Proceedings*. Springer, 2005, vol. 3794.

[6] V. I. GmbH, "Car2x –when cars are talking," 2014.

[7] A. Sari, O. Onursal, and M. Akkaya, "Review of the security issues in vehicular ad hoc networks (vanet)," *International Journal of Communications, Network and System Sciences*, vol. 8, no. 13, p. 552, 2015.

[8] R. Popescu-Zeletin, I. Radusch, and M. A. Rigani, *Vehicular-2-X communication: state-of-the-art and research in mobile vehicular ad hoc networks*. Springer Science & Business Media, 2010.

[9] D. Corporation. Bluetooth Technology for Warehouse Operations. https://www.datexcorp.com/bluetooth-technology-for-warehouse-operations/.

[10] T. Agarwal. Wireless communication with transceivers. https://www.efxkits.co.uk/role-of-2-4-ghz-wireless-communication-transceivers-in-communication/.

[11] N. S. Compulink. Wi-Fi Wireless Network Service. http://compulinklb.com/networking/.

[12] J. Tieu and S. Ye, "Wi-fi direct services," 2014.

[13] T. Instruments. (2011) OMAP Wireless Connectivity NLCP WiFi Direct Configuration Scripts. http://processors.wiki.ti.com/index.php/OMAP_Wireless_Connectivity_NLCP_WiFi_Direct_Configuration_Scripts.

[14] A. GARCIA-SAAVEDRA and P. SERRANO, "Device-to-device communications with wifi direct: Overview and experimentation," *IEEE Wireless Communications*, p. 97, 2013.

[15] V. G. Gunda, "Exploiting wi-fi-direct service discovery for prototyping of car-to-car communication," 2016.

[16] alphr. Wi-Fi. http://www.alphr.com/features/367675/how-wi-fi-works.

[17] W. Alliance, "Wi-fi peer-to-peer (p2p) technical 7 specification," *www. wi-fi. org/Wi-Fi_Direct. php*.

[18] R. Bergelt, M. Vodel, and W. Hardt, "Energy efficient handling of big data in embedded, wireless sensor networks," in *Sensors Applications Symposium (SAS), 2014 IEEE*. IEEE, 2014, pp. 53–58.

[19] A. M. Orozco, R. Michoud, and G. Llano, "Routing protocols simulation for efficiency applications in vehicular environments," *Sistemas & Telemática*, vol. 11, no. 27, pp. 27–42, 2013.

[20] R. Bergelt, "Optimierungsverfahren zur steigerung der energieeffi zienz in drahtlosen kommunikationsszenarien," 2012, pp. 49–52.

[21] V. Abinayaa and A. Jayan, "Case study on comparison of wireless technologies in industrial applications," *International Journal of Scientific and Research Publications*, vol. 4, no. 2, pp. 1–4, 2014.

[22] C. Satish, *Inter-vehicular communication for collision avoidance using wi-fi direct*. Rochester Institute of Technology, 2014.

[23] R. M. Siegers, "Optimizing traffic flows with data communication," *commerce germany OFFICIAL PUBLICATION OF THE AMERICAN CHAMBER OF COMMERCE IN GERMANY*, 2015.

[24] R. Bauza, J. Gozalvez, and J. Sanchez-Soriano, "Road traffic congestion detection through cooperative vehicle-to-vehicle communications," in *Local Computer Networks (LCN), 2010 IEEE 35th Conference on*. IEEE, 2010, pp. 606–612.

[25] K. Mori, O. Shagdar, S. Matsuura, M. Tsukada, T. Ernst, and K. Fujikawa, "Experimental study on channel congestion using ieee 802.11 p communication system," in *IPSJ Technical Workshop on Mobile Computing and Ubiquitous Communications*, 2013.

[26] A. Mellouk, S. Fowler, B. Daachi, and S. Hoceini, *Wired/Wireless Internet Communications: 12th International Conference, WWIC 2014, Paris, France, May 26-28, 2014, Revised Selected Papers*. Springer, 2014, vol. 8458.

[27] H.-M. Tsai, W. Viriyasitavat, O. K. Tonguz, C. Saraydar, T. Talty, and A. Macdonald, "Feasibility of in-car wireless sensor networks: A statistical evaluation," in *Sensor, Mesh and Ad Hoc Communications and Networks, 2007. SECON'07. 4th Annual IEEE Communications Society Conference on*. IEEE, 2007, pp. 101–111.

[28] B. Nagar, "Efficient handling of big data analytics in densely distributed sensor networks," 2015.

[29] S. Blokzyl, M. Vodel, and W. Hardt, *A Hardware-accelerated Real-time Image Processing Concept for High-resolution EO Sensors*. Deutsche Gesellschaft für Luft-und Raumfahrt-Lilienthal-Oberth eV, 2012.

[30] M. Rezaei, M. Sarshar, and M. M. Sanaatiyan, "Toward next generation of driver assistance systems: A multimodal sensor-based platform," in *Computer and Automation Engineering (ICCAE), 2010 The 2nd International Conference on*, vol. 4. IEEE, 2010, pp. 62–67.

[31] K.-H. Lee, J.-H. Kim, and S. Cho, "Power saving mechanism with network coding in the bottleneck zone of multimedia sensor networks," *Computer Networks*, vol. 96, pp. 58–68, 2016.

[32] K. Prasanna and M. Thungamani, "Secured query processing in wireless sensor networks," *International Journal of Engineering Innovations and Research*, vol. 2, no. 6, p. 535, 2013.

[33] C. Jehan and D. S. Punithavathani, "Cluster based trustable target detection and tracking scheme for wireless sensor networks."

[34] C. C. Aggarwal, *Managing and mining sensor data*. Springer Science & Business Media, 2013.

[35] A. R. Nafees and S. B. Deshmukh, "Performance evaluation of pegasis and ieepb routing protocols in wireless sensor networks," *Performance Evaluation*, vol. 3, no. 4, 2015.

[36] A. Bagula, "Applications of wireless sensor networks," 2010.

[37] F. Zidat, J.-P. Lecointe, F. Morganti, J.-F. Brudny, T. Jacq, and F. Streiff, "Non invasive sensors for monitoring the efficiency of ac electrical rotating machines," *Sensors*, vol. 10, no. 8, pp. 7874–7895, 2010.

[38] J. Sen, "Security in wireless sensor networks," *Wireless Sensor Networks: Current Status and Future Trends*, vol. 407, 2012.

[39] R. Rosemark, W.-C. Lee, and B. Urgaonkar, "Optimizing energy-efficient query processing in wireless sensor networks," in *Mobile Data Management, 2007 International Conference on.* IEEE, 2007, pp. 24–29.

[40] Y. Yao, J. Gehrke *et al.*, "Query processing in sensor networks." in *Cidr*, 2003, pp. 233–244.

[41] R. Balaji and V. Bhootna, "Query processing in wireless sensor network."

[42] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "Tinydb: an acquisitional query processing system for sensor networks," *ACM Transactions on database systems (TODS)*, vol. 30, no. 1, pp. 122–173, 2005.

[43] M. Vodel, "Energieeffiziente kommunikation in verteilten, eingebetteten systemen," 2013.

[44] E. Srie Vidhya Janani and P. Ganesh Kumar, "Energy efficient cluster based scheduling scheme for wireless sensor networks," *The Scientific World Journal*, vol. 2015, 2015.

[45] T. U. C. Chair of Computer Engineering. Planet - platform for ambient networking. https://www.tu-chemnitz.de/informatik/ce/research/planet.php.

[46] rzheng, "Wireless sensor networks and vehicular networks," 2013.

[47] G. N. RAO and M. MARUTI, "Improve the efficiency and security of wireless sensor networks for handling the big data based on embedded techniques," 2015.

[48] A. Tomar, "Introduction to zibgbee technology," vol. 1, 2011.

[49] P. Saraswathi and M. Prabha, "A novel approach for automatic monitoring of power consumption using smart meter," 2015.

[50] V. Custodio, F. J. Herrera, G. López, and J. I. Moreno, "A review on architectures and communications technologies for wearable health-monitoring systems," *Sensors*, vol. 12, no. 10, pp. 13 907–13 946, 2012.

[51] C. Jin, J.-W. Choi, W.-S. Kang, and S. Yun, "Wi-fi direct data transmission for wireless medical devices," in *Consumer Electronics (ISCE 2014), The 18th IEEE International Symposium on.* IEEE, 2014, pp. 1–2.

[52] D. Camps-Mur, A. Garcia-Saavedra, and P. Serrano, "Device-to-device communications with wi-fi direct: overview and experimentation," *IEEE wireless communications*, vol. 20, no. 3, pp. 96–104, 2013.

[53] Y. Duan, "Cooperative data transfers for 5g networks," Ph.D. dissertation, Politecnico di Torino, 2017.

[54] R. Baumann, "Vehicular ad hoc networks (vanet):(engineering and simulation of mobile ad hoc routing protocols for vanet on highways and in cities)," Ph.D. dissertation, ETH, Swiss Federal Institute of Technology Zurich [Department of Computer Science, Computer Systems Institute], 2004.

[55] R. Barskar and M. Chawla, "Vehicular ad hoc networks and its applications in diversified fields," *International Journal of Computer Applications*, vol. 123, no. 10, 2015.

[56] G. Karagiannis, O. Altintas, E. Ekici, G. Heijenk, B. Jarupan, K. Lin, and T. Weil, "Vehicular networking: A survey and tutorial on requirements, architectures, challenges, standards and solutions," *IEEE communications surveys & tutorials*, vol. 13, no. 4, pp. 584–616, 2011.

[57] A. Roberto and A. Gulielmo, "Wave: a tutorial," *IEEE Communication Magazine*, vol. 47, pp. 126–133, 2009.

[58] C. E. Perkins and P. Bhagwat, "Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers," in *ACM SIGCOMM computer communication review*, vol. 24, no. 4. ACM, 1994, pp. 234–244.

[59] Z. J. Haas, M. R. Pearlman, and P. Samar, "The zone routing protocol (zrp) for ad hoc networks," 2002.

[60] A. Mukhija and R. Bose, "Reactive routing protocol for mobile ad-hoc networks," *Department of Mathematics Indian Institute of Technology*, 2001.

[61] M. Torrent-Moreno, F. Schmidt-Eisenlohr, H. Füßler, H. Hartenstein *et al.*, *Packet forwarding in VANETs, the complete set of results.* Univ., Fak. für Informatik, Bibl., 2006.

[62] S. Yousefi, M. Fathy, and A. Benslimane, "Performance of beacon safety message dissemination in vehicular ad hoc networks (vanets)," *Journal of Zhejiang University-Science A*, vol. 8, no. 12, pp. 1990–2004, 2007.

[63] P. Singh, D. Singh, and V. Singh, "Evaluation of routing protocols in manets with varying network scope," in *International Conference on Information and Network Technology, IACSIT Press, Singapore*, 2012, pp. 215–219.

[64] Y. Sakurai and J. Katto, "Aodv multipath extension using source route lists with optimized route establishment," in *Wireless Ad-Hoc Networks, 2004 International Workshop on.* IEEE, 2004, pp. 63–67.

[65] G. RESMINI, "Opportunistic wi-fi direct networking with channel state based routing algorithm," 2015.

[66] W.-F. P. Setup. https://en.wikipedia.org/wiki/Wi-Fi_Protected_Setup.

TECHNISCHE UNIVERSITÄT CHEMNITZ

Studentenservice – Zentrales Prüfungsamt

Selbstständigkeitserklärung

Name: Jitta Vorname: Srinivasu geb. am: 20.08.1991 Matr.-Nr.: 357928	**Bitte beachten:** 1. Bitte binden Sie dieses Blatt am Ende Ihrer Arbeit ein.

Selbstständigkeitserklärung*

Ich erkläre gegenüber der Technischen Universität Chemnitz, dass ich die vorliegende Masterarbeit selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe.

Die vorliegende Arbeit ist frei von Plagiaten. Alle Ausführungen, die wörtlich oder inhaltlich aus anderen Schriften entnommen sind, habe ich als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch nicht als Prüfungsleistung eingereicht und ist auch noch nicht veröffentlicht.

Datum: 07.02.2018 Unterschrift: ……

* Statement of Authorship

I hereby certify to the Technische Universität Chemnitz that this thesis is all my own work and uses no external material other than that acknowledged in the text.

This work contains no plagiarism and all sentences or passages directly quoted from other people's work or including content derived from such work have been specifically credited to the authors and sources.

This paper has neither been submitted in the same or a similar form to any other examiner nor for the award of any other degree, nor has it previously been published.